D 2018

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

IC PROTECTION AGAINST JTAG/IJTAG-BASED ATTACKS

XUANLE REN
TESE DE DOUTORAMENTO APRESENTADA
À FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO EM
ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

IC PROTECTION AGAINST JTAG/IJTAG-BASED ATTACKS

TESE SUBMETIDA PARA OBTENÇÃO DO

GRAU DE DOUTOR

EM

ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Xuanle Ren

ORIENTADORES: PROFESSOR SHAWN BLANTON, PROFESSOR VÍTOR GRADE TAVARES

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

PORTO, PORTUGAL

DEZEMBRO 2018

Resumo

A segurança em circuitos integrados (CIs) está, presentemente, a tornar-se num desafio bem estabelecido. Vários tipos de ataques a CIs têm vindo a ser reportados, incluindo casos de engenharia reversa, carregamento de dados no CI e controlo/modificação da operação do CI. A norma IEEE 1149.1, vulgarmente conhecida por *Joint Test Action Group* (JTAG), providencia o acesso de teste a um CI. A JTAG é principalmente utilizada em CIs para teste e depuração, mas também é utilizada em campo (no utilizador), o que permite o acesso pós-fabricação a sub-sistemas internos do CI. Desde a sua introdução, e em consequência da necessidade de endereçar uma série de desafios específicos que o teste e depuração de CIs coloca, que a norma IEEE 1149.1 tem vindo a ser continuamente expandida a uma variedade de outras normas. Por exemplo, a norma IEEE 1687 facilita, de forma eficiente, o acesso a instrumentos embutidos pela disponibilização de redes de examinação (*scan networks*) reconfiguráveis. No entanto, uma vez que a JTAG é deixada intacta e operacional após o fabrico do CI, inevitavelmente disponibiliza uma "porta indireta" que pode ser explorada fora do seu uso pretendido. Naturalmente, e pelas razões expostas, a JTAG tem vindo a ser adotada de forma universal, pelo que a utilização desta como porta de acesso indiscreto tem ganhado uma atenção crescente.

Há várias abordagens propostas anteriormente para impedir o acesso ilegítimo ao JTAG, dentre elas, a fusão da porta JTAG e o acesso encriptado são exemplos frequentes. No entanto, a fusão das portas da JTAG desativa permanentemente a depuração após o fabrico, enquanto a encriptação do acesso tipicamente está sujeita à interceção das chaves ou requer a ligação a uma rede de dados. Neste trabalho é desenvolvido um método inovador, baseado em *machine learning* (ML), para a monitorização e deteção de acesso ilegítimo à JTAG. O detetor desenvolvido pode ainda ser implementado em CI, de tal forma que estes ataques possam ser detetados e impedidos rapidamente. Comparando com métodos anteriores, a deteção de ataques com base em métodos de ML não só preservam a funcionalidade de depuração/programação em campo, como também minimizam o risco de interceção das chaves de acesso. Para mais, o método proposto, sendo ortogonal

aos anteriores, pode ser combinado com eles (por exemplo, com encriptação) para uma proteção complementar da JTAG.

A construção de um modelo de ML capaz de distinguir acessos legítimos ou ilegítimos à JTAG requer: 1) uma coleção de características (*features*) relevantes, 2) uma grande quantidade de dados com qualidade e 3) um algoritmo de ML eficaz. A operação JTAG de vários sistemas é estudada, incluindo os processadores OpenSPARC T1, T2 e o Leon, de forma a que as características mais relevantes sejam encontradas. De seguida, são recolhidos os dados correspondentes às características relevantes e resultantes do estudo da operação legítima da JTAG, bem como dos respetivos ataques aplicáveis. Para a deteção de ataques, são desenvolvidos e avaliados detetores baeados em janelas deslizantes e modelos de sequência. Para o primeiro, a operação da JTAG é segmentada em subsequências de tamanho fixo que são posteriormente classificadas por um modelo de ML. As experiências, com base em classificadores de árvores de decisão, apresentam um erro de 3% e 5% para a OpenSARC T1 e T2, respetivamente. Os modelos de sequência, sendo capazes de caracterizar de forma mais precisa a dependência das sequências, reduzem o erro para valores ainda mais baixos (i.e., 1% e 3%).

Após a deteção de um ataque, o acesso à JTAG é restringida. Mais especificamente, é desenvolvida uma arquitetura JTAG que não só desativa a observabilidade e controlabilidade da JTAG, como também ofusca a sua operação após o desencadear da restrição. Isto evita que o atacante possa obter os parâmetros do modelo de ML por engenharia reversa, tornando o detetor robusto a ataques de adversário. Experiências com o OpenSPARC T2 mostram que a modificação proposta da arquitetura da JTAG apenas adiciona 22% na área de CI, isto quando comparada com a JTAG original, e a adição global, face à OpenSPARC T2, são uns meros 0.01%.

O método de deteção de ataque é posteriormente expandido a redes IJTAG. O desafio na deteção de ataques a IJTAGs reside na elevada dimensionalidade dos dados recolhidos durante a operação da IJTAG. De forma a tornar possível uma deteção em CI, a elevada dimensionalidade dos dados é comprimida com recurso a uma matriz de *low-density parity-check* (LDPC). De acordo

com as experiências realizadas numa versão modificada do processador OpenSPARC T2, a utilização de compressão com PDPC elemina 91% das características e reduz o tamanho do circuito final em 43%, sem afetar a qualidade da deteção.

Abstract

Security is now becoming a well-established challenge for integrated circuits (ICs). Various types of IC attacks have been reported, including reverse engineering, dumping on-chip data, and controlling/modifying IC operation. IEEE Standard 1149.1, commonly known as Joint Test Action Group (JTAG), is a standard for providing test access to an IC. JTAG is primarily used at the time of IC fabrication for test and debug, but is also employed in the field, allowing access to internal sub-systems of the IC. Since introduced, IEEE Standard 1149.1 has been continuously extended to a variety of standards for addressing challenges in IC test and debug. For instance, IEEE Standard 1687 facilitates efficient access to embedded instruments by supporting reconfigurable scan networks. Because JTAG is left intact and operational after fabrication, it inevitably provides a "backdoor" that can be exploited outside its intended use. This backdoor has gained increasing attention due to the ubiquitous adoption of the JTAG.

Previous approaches, such as fusing-off the JTAG port and encrypting JTAG access, have been proposed to prevent illegitimate access to the JTAG. However, fusing-off the JTAG also permanently disables in-field debugging, while encrypting JTAG access typically suffers from key eavesdropping or requires availability of a network. In this work, a novel, machine-learning (ML) based method is developed for monitoring and detecting illegitimate access to the JTAG. The ML-based detector can be further implemented on chip so that these attacks can be detected and prevented quickly. Compared to previous work, the ML-based attack detection not only preserves the functions of in-field debugging/programming, but also minimizes the risk of key eavesdropping. Moreover, the proposed method, orthogonal with previous ones, can be combined with them (such as encryption) to provide complementary protection for the JTAG.

Constructing an ML model that distinguishes legitimate and illegitimate JTAG access requires 1) relevant features, 2) a large amount of high-quality data, and 3) an effective ML algorithm. To derive relevant features, JTAG operation of various designs are studied, including the OpenSPARC T1, T2 and the LEON3 processors. Then, data corresponding to the features are collected through

studying legitimate JTAG operation and attacks applicable to the JTAG. To detect attacks, sliding-window-based detectors and sequence models are developed and evaluated. For sliding-window-based detectors, JTAG operation is segmented into fixed-size sub-sequences which are then classified by an ML model. Experiments using a decision-tree classifier demonstrate an error rate of 3% and 5%, for the OpenSPARC T1 and T2, respectively. Further, sequence models, able to characterize serial dependence more accurately, reduce the error rate to even lower levels (i.e., 1% and 3%).

Upon detection of an attack, access to the JTAG is restricted. Specifically, a secure JTAG architecture is developed, which can not only disable observability and controllability provided by the JTAG, but also obfuscate the JTAG operation triggering the restriction. This prevents the attacker from reverse engineering the parameters of the ML model, making the detector robust to adversarial attacks. Experiments using the OpenSPARC T2 show that the modification of the JTAG architecture only adds 22% chip area compared to the original JTAG, and the overall addition to the OpenSPARC T2 is mere 0.01%.

The method of attack detection is further extended to IJTAG networks. A challenge of detecting IJTAG attacks resides in the high-dimensional data collected from IJTAG operation. To make on-chip detection possible, the high-dimensional data is compressed using a low-density parity-check (LDPC) matrix. According to the experiments using a modified version of the OpenSPARC T2 processor, the use of LDPC-based compression eliminates 91% of the features, and reduces circuit size by 43% without affecting detection accuracy.

Acknowledgement

I would like to give the first and deepest thank to my advisors, Prof. Shawn Blanton and Prof. Vítor G. Tavares, without whose continuous support and guidance, I am not able to complete this Ph.D. dissertation. During the six-year co-working, I have learned from them how to keep motivated and do relevant research. Moreover, I see them as mentors because of their wisdom and attitude towards not only scientific research, but also every aspect of the life. They will be excellent examples that enlighten my future career.

Second, I would like to acknowledge Prof. Diana Marculescu, Prof. Jeff Schneider, Prof. Kenneth Mai, and Prof. Jaime Cardoso, for sitting in my defense committee. I am grateful that they take time from busy schedule to follow up with my work. Their insights and expertise contribute to outlining and improving this dissertation.

I would also like to thank my colleagues and friends for supporting me through these years. In particularly, I am grateful to the ACTL members: Matthew Beckler, Ben Niewenhuis, Yang Xue, Cheng Xue, Xiang Lin, Zeye Liu, Soumya Mittal, Phillip Fynan, Brian Osbun, Shanghang Zhang, Qicheng Huang, Chenlei Fang, Alfred Nguyen, and Jaime Kang. They are wonderful and smart people that have inspired my research a lot. I would also like to thank the co-authors of my publications: Francisco P. Torres and Mitchell Martin. In addition, I would like to thank the colleagues in Porto: Ganga Bahubalindruni, Iman Kianpour and Helder Avelar. They helped me so much with both research and life in Portugal.

Next, I would like to give special thanks to my family, the source of my hope and power. Their encouragement and support make me overcome difficulties and become a better person. I would also like to thank all my friends in Pittsburgh, in Porto, and in China, for sharing incredible and enjoyable moments with me.

Last but not least, I would like to acknowledge the financial support that makes this Ph.D. work possible. In particular, I would like to thank Portuguese Foundation for Science and Technology for

granting me a five-year scholarship (SFRH/BD/52166/2013) through the Carnegie Mellon Portugal Program. I would also like to express my gratitude to Prof. Shawn Blanton for granting me financial support for the last year of my Ph.D.

Contents

Resumo iii

Abstract vi

Acknowledgement viii

List of Tables xiv

List of Figures xvi

1 Introduction **1**

1.1 Characterizing JTAG/IJTAG Operation 10

1.2 Detecting JTAG/IJTAG Attacks 12

1.3 Restricting JTAG Access 14

1.4 Dissertation Organization 15

2 Characterization of JTAG Operation **17**

2.1 JTAG Architecture 17

2.1.1 Standard JTAG Architecture 18

2.1.2 Bus-based Debug Architecture 20

2.2 JTAG Attacks 23

2.3 Feature Characterization 26
2.4 Experiments 29
2.4.1 OpenSPARC T1 and T2 29
2.4.2 LEON3 33
2.5 Summary 34

3 Detection of JTAG Attacks **36**
3.1 Detection Using a Sliding Window 37
3.2 Classification Models 40
3.2.1 Binary Classifiers 40
3.2.2 Recurrent Sliding Window 43
3.2.3 One-class Models 43
3.3 Delayed Labeling 45
3.4 Sequence Models 49
3.4.1 Recurrent Neural Network 49
3.4.2 Hidden Markov Model 51
3.5 Detection of Unknown Attacks 53
3.6 Experiments 56
3.6.1 Search for Window Size 56
3.6.2 Sliding-window-based Detectors 56
3.6.3 Sequence Models 60
3.6.4 Detection of Unknown Attacks 62
3.7 Discussion 64

3.8 Summary 69

4 Detection of IJTAG Attacks **70**

4.1 Background 71

4.1.1 IJTAG Architecture 71

4.1.2 Prior Work 74

4.2 IJTAG Attack Detection 75

4.2.1 IJTAG Attacks 75

4.2.2 Feature Extraction 76

4.2.3 Compressed Sensing 77

4.2.4 Overall Flow 80

4.3 Experiments 82

4.3.1 Modification of OpenSPARC T2 82

4.3.2 Evaluation of LDPC Matrices 84

4.3.3 Evaluation of IJTAG Attack Detection 84

4.4 Discussion 85

4.5 Summary 87

5 JTAG Restriction and Detector Implementation **89**

5.1 Restriction of JTAG Access 90

5.2 Security Analysis 93

5.2.1 Adversarial Attack 93

5.2.2 Disguised Attack 95

5.2.3 Other Security Concerns 97
5.3 Detector Implementation 97
5.3.1 ML Classifiers 98
5.3.2 Evidence Collector 101
5.3.3 Overhead Compared to Software 102
5.4 Discussion 105
5.5 Summary 107

6 Summary and Future Work **108**
6.1 Dissertation Contribution 109
6.2 Future Work 112

A Adversarial Analysis **115**

Glossary **118**

Bibliography **121**

List of Tables

1.1 Examples of industrial solutions to securing the JTAG. 5

1.2 Attack model assumed in this work (the entries with ✓ means that the attacker has the corresponding ability, while ✗ means the attacker does not). 7

2.1 A list of various non-intrusive attacks. 23

2.2 JTAG attack strategies are organized in three main steps and nine sub-steps. 25

2.3 Some features used for characterizing JTAG operation. 27

2.4 The JTAG functions implemented in the OpenSPARC T1 and T2. . . . 30

2.5 Legitimate JTAG operation and attacks are created for the OpenSPARC T1 and T2. 30

3.1 The parameters selected for various detection models. 58

4.1 Features used for characterizing IJTAG operation. 76

4.2 Synthesis results are compared for the random forest (with three trees) detectors (Gate equivalent = Total area / Area of a two-input NAND gate). 87

4.3 THE PROPOSED APPROACH IS COMPARED TO PRIOR WORK. 88

5.1 FOR EACH TYPE OF DR, CONTROLLABILITY AND OBSERVABILITY ARE PROHIBITED THROUGH SPECIFIC ACTIONS. 90

5.2 ATTACK MODEL AIMED AT THE DETECTOR (AS A SUPPLEMENT TO TABLE 1.2). 94

5.3 DETECTORS BASED ON DIFFERENT CLASSIFIERS ARE COMPARED (GATE EQUIVALENT = TOTAL AREA / AREA OF A TWO-INPUT NAND GATE). 102

List of Figures

1.1 Example boards with the JTAG ports exposed: (a) D-Link DIR-620 router [1], and (b) Broadcom BCM5354 processor [2]. 2

1.2 The JTAG standard has been extended to a variety of IEEE standards. 3

1.3 This diagram describes the methods of detecting and restricting illegitimate JTAG accesses developed in this dissertation. 8

1.4 Supervised and unsupervised learning are illustrated using a classification and a clustering problem, respectively. (a) The goal of classification is to identify an optimal boundary between two or more classes, while (b) the goal of clustering is to group observations based on their density. 13

2.1 The block diagram of a typical JTAG architecture. 19

2.2 The JTAG TAP controller is a one-input FSM. 20

2.3 The blocks of the LEON3 are connected through an AMBA Advanced High-performance Bus (AHB) [3]. 21

2.4 The JTAG, acting as an AHB master, can access processor registers through a debug support unit (DSU) [3]. 22

2.5 The importance of features is calculated using a decision tree and mutual information for (a) the OpenSPARC T1 and (b) the OpenSPARC T2. In each chart, the results of recursive feature elimination is indicated by the order of the features, i.e., features on the right will be eliminated firstly. 31

2.6 Legitimate JTAG operation and attacks for (a) the OpenSPARC T1 and (b) the OpenSPARC T2, are reorganized using a sliding window, and then plotted using t-SNE. 32

2.7 Legitimate JTAG operation and attacks via the bus of the LEON3 processor are reorganized using a sliding window, and then plotted using t-SNE. 33

3.1 The overall flow for online detection of JTAG attacks using a sliding window ($w = 3$). 38

3.2 Illustration of some example classifiers, including (a) a decision tree, (b) a linear SVM, (c) a three-layer feedforward ANN, and (d) k-NN. 41

3.3 Sequence collected in real-time is compared with three representative sequences, namely 7-1-3, 2-3-1-4, and 8-8-7, from the first opcode to the last. Each representative sequence is associated with a local score that may increment, saturate (saturation score is 5 in this example) or reset, according to the matching result. A global score, equal to the maximum local score, serves as an indicator of the security status of the JTAG. 44

3.4 The computation of $\widetilde{\alpha}_L(t)$ and $\widetilde{\alpha}_A(t)$ involves a lattice-like process. 47

3.5 The parameters of an HMM used for collecting evidence of an attack. The hidden state always starts from S_L, i.e., assuming the user as legitimate initially. State transition, either from S_L to S_A or from S_A to S_L, occurs with a small probability v. The emission probability can be derived through analyzing the probabilistic predictions produced for legitimate operation and attacks, respectively. More precisely, legitimate operation is more likely to produce a negative prediction that results in a small o_t, while an attack is more likely to produce a positive prediction that results in a large o_t. 47

3.6 A gated recurrent unit (GRU). 50

3.7 The architecture of a two-level HHMM. 52

3.8 A binary classifier suffers from open-space risk, i.e., unable to correctly classify observations that are "far" from historically-observed data (e.g., legitimate operation and known attacks). 54

3.9 A cascade model is constructed by combining a one-class SVM and a binary SVM for detecting unknown attacks. 55

3.10 The impact of window size (w) on the regularized KL-divergence between legitimate JTAG operation and attacks, for (a) the OpenSPARC T1, (b) the OpenSPARC T2, and (c) the LEON3. 57

3.11 Sliding-window-based detectors and the representative-based anomaly detector are evaluated using five-fold cross validation, with the performance evaluated using error rate, FPR, and FNR. 59

3.12 The cost decreases as the training of an RNN proceeds. 61

3.13 Sequence models, including an RNN, an HMM, and a two-level HHMM, are evaluated using error rate, FPR, and FNR. 62

3.14 Two HMMs, namely HMM-X and HMM-Y, are trained based on legitimate JTAG operation and attacks, respectively. Each testing trace is evaluated using both HMMs, with the log-probability plotted. 63

3.15 Detection of unknown attacks that target different IC components are evaluated for various models, including (a) a decision tree, (b) the cascade model, and (c) an RNN. 65

3.16 Detection of unknown attacks that exploit different strategies (listed in Table 2.2) are evaluated for various models, including (a) a decision tree, (b) the cascade model, and (c) an RNN. 66

3.17 Legitimate JTAG operation and attacks for the OpenSPARC T2 are plotted using t-SNE. Two types of attacks (targeting an IC component not included in training) are regarded as unknown, one located in (a) an open space, and the other located in (b) an area overlapping with legitimate operation. 67

4.1 The structure of a segment insertion bit (SIB). 72

4.2 An example of an IJTAG architecture composed of four instruments, each of which is gated by a SIB and/or selected by an SCB. 73

4.3 An industrial memory macro wrapped by a six-bit TDR. 73

4.4 (a) An example of a Tanner graph with eight variable nodes and four check nodes. (b) The adjacent variable nodes and check nodes of the Tanner graph in Figure 4.4(a) are traversed starting from v_0. 79

4.5 As d (density of A) increases, g (the local girth) decreases. The largest d that satisfies $g > 4$ is chosen for constructing the LDPC matrix A. 80

4.6 The overall flow of IJTAG attack detection. 81

4.7 The OpenSPARC T2 is partitioned into 11 sub-systems, and then each sub-system is wrapped by a TDR. In addition, each TDR is gated by a SIB, and all SIBs comprise a daisy chain that is accessible via the JTAG port. 82

4.8 The density distribution (i.e., number of non-zero entries) for the collected data whose dimension is 280. 83

4.9 The performance of reconstructing the original data is evaluated for LDPC matrices and three other types of matrices. 85

4.10 The data, whose dimension is reduced by (a) the LDPC-based feature reduction and (b) feature selection using a decision tree, are classified using a random forest. 86

5.1 To protect the JTAG, access to DRs is modified as shown in red. 91

5.2 The error rate of identifying disguised attacks is evaluated using a decision-tree detector and compared with the error rate of identifying legitimate operation and undisguised attacks. An attack trace is disguised through segmenting the trace and then inserting legitimate sequences between the segments. 96

5.3 The overall architecture of the sliding-window-based detector. 98

5.4 Architecture of (a) a combinational tree, (b) a sequential tree, and (c) an SVM. . . . 99

5.5 Architecture of the evidence collector. 101

5.6 Area of the SVM detector and the mean squared error of the classification results (compared to software) are impacted by the number of bits for α_i (n_alpha_bit) and the RBF values (n_rbf_bit). 103

5.7 Performance and size of (a) the random-forest-based detector and (b) the SVM-based detector are impacted by the variety of attacks used in classifier training. . . 106

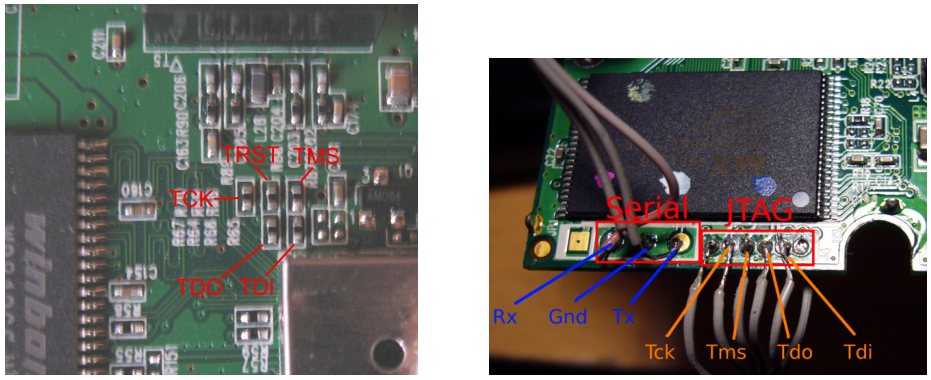
6.1 A combination of hardware and software for attack detection. 113

6.2 A mechanism that permits legitimate users to re-enable access to the JTAG. 113

Chapter 1

Introduction

During the past decades, integrated circuits (ICs) have been widely used in various fields, from consumer devices to military equipment, because of the powerful capability of computation they provide. Meanwhile, security of ICs is becoming a central issue considered by both IC providers and users. An IC, if not secured, may be attacked by hackers, which further threatens the security of personal information, IC providers and users, and even military defense. To ensure the security of ICs, one typically needs to analyze potential vulnerabilities, and then develop solutions to mitigate them. Today, the supply chain of ICs, including design, fabrication, testing, integration, packaging, and shipping, requires collaboration of various participants across the globe. This makes the security problem of ICs even more severe and more complex. Any untrusted participant could be an adversary and attack the IC using various tools and techniques. For example, credential on-chip information can be dumped via the ports of the IC [4, 5] and through side-channel signals, such as power consumption [6], electromagnetic (EM) emissions [7] and the testing infrastructure [8–15]. On-chip information can also be dumped using hardware Trojans [16] that take the form of a modification of the original circuitry. Hardware Trojans can also bypass/disable the security characteristics of a system [17], and even destroy the IC [18]. Other than on-chip information, intellectual property (IP) cores are also a common attack target. An attacker can reverse engineer an IP core intrusively (via chemical etching [19]) or non-intrusively (via optical imag-



(a) **(b)**

Figure 1.1: Example boards with the JTAG ports exposed: (a) D-Link DIR-620 router [1], and (b) Broadcom BCM5354 processor [2].

ing [20] and X-ray reconstruction [21]). An IP core, if stolen, can be used for overproducing or cloning more copies without claiming the ownership of the IP, thus causing significant profit loss for the IP owner. Another threat to ICs involves counterfeiting. An IC is counterfeit if it (or a part of it) is recycled, remarked, cloned, or known to be defective [22]. IC counterfeiting causes a significant profit loss ($100 billion annually [23]) and reputation damage for legitimate companies. Finally, because there exist various attacking scenarios, to develop countermeasures, one needs to make assumptions of who could be the adversary and which tools/strategies the adversary might exploit.

Among the many security challenges that exist, the vulnerabilities caused by the Design-for-Testability (DFT) infrastructure has gained increasing attention. DFT aims to make manufacturing test of ICs easier and more efficient by adding testability features to the IC. Among the various DFT techniques, the standard test access port and boundary scan architecture, defined by the IEEE Standard 1149.1 (also referred to as JTAG), is a widely-used infrastructure that enhances IC manufacturing test by providing a serial communication interface for accessing internal subsystems of an IC [24]. Figure 1.1 shows examples of JTAG ports that are found on printed-circuit boards, including D-Link DIR-620 router [1] and Broadcom BCM5354 processor [2]. When being tested, an IC is configured in a test mode where the flip-flops within the IC are connected as one or multiple shift registers, typically referred to as scan chains. Scan chains can then be used for loading

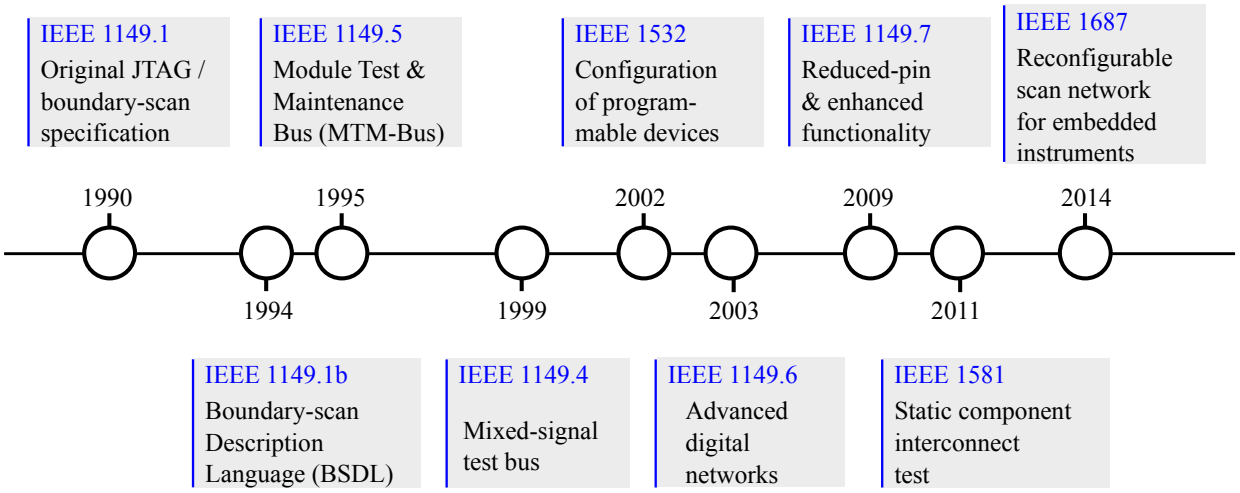


Figure 1.2: The JTAG standard has been extended to a variety of IEEE standards.

test stimuli into the IC and capturing test responses from the IC. Since introduced as an industry standard in the 1990s, JTAG continues to grow in terms of capability and usefulness. Today, JTAG is almost implemented within all ICs, not only for manufacturing test, but also for in-field debugging and programming [25–27]. For example, the JTAG within many processors provides access to debugging registers (e.g., the trace buffer register), which is critical for software and firmware debugging [28, 29]. The JTAG of the OpenSPARC T2 processor can be used for accessing cache, configuring control registers, and programming electronic fuses (eFuse) [30].

To solve the problems arising in testing increasingly-complex ICs, 1149.1 has been extended to a variety of new IEEE standards, some of which are shown in Figure 1.2. For example, IEEE Standard 1149.4 expands boundary scan to analog and mixed-signal testing [31], while IEEE Standard 1500 supports the test of embedded cores by facilitating test integration and test reuse [32]. Among the extensions, IEEE Standard 1687, also referred to as internal JTAG (IJTAG), facilitates efficient access to embedded instruments[1] [33]. IJTAG reduces the cost and difficulty of testing individual IP cores, and is therefore becoming more and more popular for testing complex ICs. It is widely believed that JTAG and its related standards will continue to be extended to address future challenges in IC test and debug [34].

[1] An instrument refers to an IP core with an IJTAG-compliant interface.

Because JTAG (or some portion of it) is usually left operational for in-field use, it inevitably provides a "backdoor" that can be utilized for accessing internal subsystems of the IC. If not protected, the JTAG can be exploited outside its intended use. Even though attackers might exploit various tools and techniques, it is reasonable to assume that JTAG will be a preferred conduit due to its ubiquitous adoption and powerful debugging functionalities for control and observation. Note that having access to the JTAG ports also enables observation of other side-channel signals (e.g., power consumption). Nevertheless, side-channel analysis is more expensive to do than just controlling/observing the JTAG ports. To exploit the JTAG, an attacker first must identify the JTAG ports on the board. Then the attacker can explore the secrets within the system, through loading data into the chip and observing chip responses from the input data. Actually, it has been demonstrated that an attacker can modify or sniff the firmware using the JTAG [26, 27, 35, 36]. A well-known example involves the hack of Xbox 360 gaming consoles. The hacker runs illegitimate code with a hacked version of firmware [37]. Even if a new version of bug-free firmware is updated, the user can still use the JTAG to downgrade the firmware to the hacked version. Another example involves illegitimate access to FPGAs. The configuration bitstream, which contains the IP information of a reconfigurable design, is mostly programmed via the JTAG. Obtaining access to the JTAG, the attacker could modify the configuration of the system, and sniff configuration bits or even retrieve the IP information. It has been reported that an FPGA for military use was attacked successfully [25]. Another well-known attack via the JTAG involves deriving the keys of on-chip cryptographic modules [8–15]. A cryptographic module typically uses a key to encrypt plaintext. However, if the flip-flops storing the intermediate results of the encryption reside in scan chains, then the attacker could dump this sensitive information using the JTAG, in order to discover the key. More precisely, every time encryption executes for one iteration in the normal mode, the attacker switches the IC to the test mode and shifts out the contents of scan chains. Repeating this process, the attacker can uncover the key through differential analysis.

Recently, the vulnerability of JTAG has received even more attention in the area of Internet-of-Things (IoT) [38]. A typical IoT network includes a number of technology components, including

TABLE 1.1: EXAMPLES OF INDUSTRIAL SOLUTIONS TO SECURING THE JTAG.

Company	Product	Solution
Intel	Sixth and seventh motherboard (e.g., 100 Series Chipset Family)	Encrypt confidential JTAG instructions; the key is distributed through a non-disclosure agreement [40, 41]
Intel	40-, 28- and 20-nm FPGAs (e.g., Arria V, Cyclone V)	Define multiple JTAG access and a secure mode that allows only mandatory JTAG instructions [42]
Intel (Altera)	Cyclone III LS FPGAs	Allow only mandatory JTAG instructions; a reset can gain full access, but erases credential data [43]
Freescale	i.MX6 series processors	Define three security modes (i.e., disabled, enabled, and boundary-scan-only) using OTP eFuse [44]
Silicon Labs	EM35x chips	Define bits in firmware that can enable a read/write protection [45]
Infineon	AURIX family controllers	Define bits in flash memory that can secure the JTAG; the keys are configured using a pre-defined instruction [46]
Samsung	ARTIK IoT platform	Contact Samsung for obtaining the key for accessing the JTAG [47]
Xilinx	Zynq-7000 AP SoC	The JTAG is disabled by default, and can be enabled/disabled through eFuse [48]
Texa Instruments	MSP families	Use several security solutions, including physical fuse, eFuse, and a JTAG lock mechanism [49]

Bluetooth, Wi-Fi, device firmware, services, APIs, a variety of network protocols, and a whole host of user-level applications. Because an IoT network strongly relies on over-the-air communication among interconnected devices, an untrusted device, connected within the network, may leave a "backdoor" that threatens not only the device, but also the entire IoT network [38, 39].

Due to these arising attacks via the JTAG, the industry has come to understand the necessity and urgency to develop countermeasures. Different from software whose security can be improved through patching, hardware cannot be patched after fabrication. Instead, the security of hardware needs to be considered during the design and manufacturing stages. Table 1.1 gives some industrial

examples of securing the JTAG. These solutions can be summarized as follows. An intuitive solution involves disabling the JTAG through one-time programmable (OTP) electronic fuse (eFuse) after manufacturing test. The use of an eFuse disables JTAG access permanently, thus preventing in-field debugging which may be a significant shortcoming for many end users [50]. Access to the JTAG can also be hindered by distributing its ports across the chip (or board), however it has been demonstrated that the JTAG ports can still be identified using exhaustive search [26]. This identification can even be accelerated using tools such as the JTAGulator [51] and SEGGER J-Link [52]. Third, on-chip compression/compaction, originally aimed at expediting manufacturing test, also protects on-chip data from being directly observed through the inherent obfuscation of test responses performed by this form of DFT [53]. However, compression/decompression only obfuscates scan-chain data, and not other JTAG functions [13]. Another commonly-used solution involves password-based authentication that requires a correct password to gain access to the JTAG [54–56]. The password is typically programmed through OTP eFuse or stored in firmware. Nevertheless, since the password needs to be distributed to legitimate users by some means, it might be eavesdropped during the distribution; this shortcoming is exacerbated if all fabricated instances share the same password. The risk of plain-text passwords can be mitigated using more complex encryption algorithms, such as DES [8], AES [9], RSA [10], and ECC [11], through a challenge-response protocol. For example, when AES (Advanced Encryption Standard) is used for encryption, the user needs to send a challenge to the chip, and the chip computes a hash using both the challenge and a unique key within the chip. The computation result, as response to the challenge, is then sent to a trusted server. Only if the response computed by the chip matches the record stored in the trusted server, then the user gains a communication session with the device. This challenge-response-based authentication provides high security, but it requires a trusted server to manage the multi-stage authentication between the user and the device, which relies on availability of a network.

To mitigate the shortcomings of prior countermeasures, new methods are developed in this dissertation that provide an orthogonal layer of protection for the JTAG. Particularly, a detector

TABLE 1.2: ATTACK MODEL ASSUMED IN THIS WORK (THE ENTRIES WITH ✓ MEANS THAT THE ATTACKER HAS THE CORRESPONDING ABILITY, WHILE ✗ MEANS THE ATTACKER DOES NOT).

The adversary can access ...	The adversary knows ...
✓ ports of the IC (including the JTAG ports)	✓ mandatory JTAG instructions
✗ side channels (e.g., power consumption, EM emission)	✗ private test/debug functions
✗ netlist of the IC	✗ JTAG instructions and their opcode
✗ affect IC fabrication (e.g., insert Trojans)	

can be implemented on chip, which monitors JTAG operation and predicts whether the user is legitimate. The motivation resides in the fact that, based on the JTAG attacks reported in the literature thus far, an attacker knows the mandatory JTAG instructions (defined by the 1149.1 standard and its related standards), but typically does not know which private test/debug functions have been implemented and which JTAG instructions have been defined to support these functions. The attack model is described in Table 1.2. In this dissertation, we further assume that the attacker only has access to the external ports of an IC, including the JTAG ports. In other words, intrusive attacks, such as obtaining the netlist of the IC or reverse engineering the design using chemical etching, are beyond the scope of this dissertation. To exploit the private JTAG functions for further exploration of the internal system, an attacker typically makes guesses about the JTAG functions according to his/her experience, and then tries to uncover them using trial-and-error strategies [36, 57]. The attacker, with malicious objectives and less knowledge about the private JTAG functions, is likely to operate the JTAG differently from a legitimate user, which provides an opportunity of detecting illegitimate access through analyzing how the JTAG is operated.

There exist some methods for detecting illegitimate JTAG operation, such as monitoring the number of clock cycles that a scan chain is shifted, checking if the user attempts to load an undefined instruction opcode, and checking if the authentication fails too many times [58]. Although these methods provide barriers for the attacker, they serve only as sanity checks, and therefore are

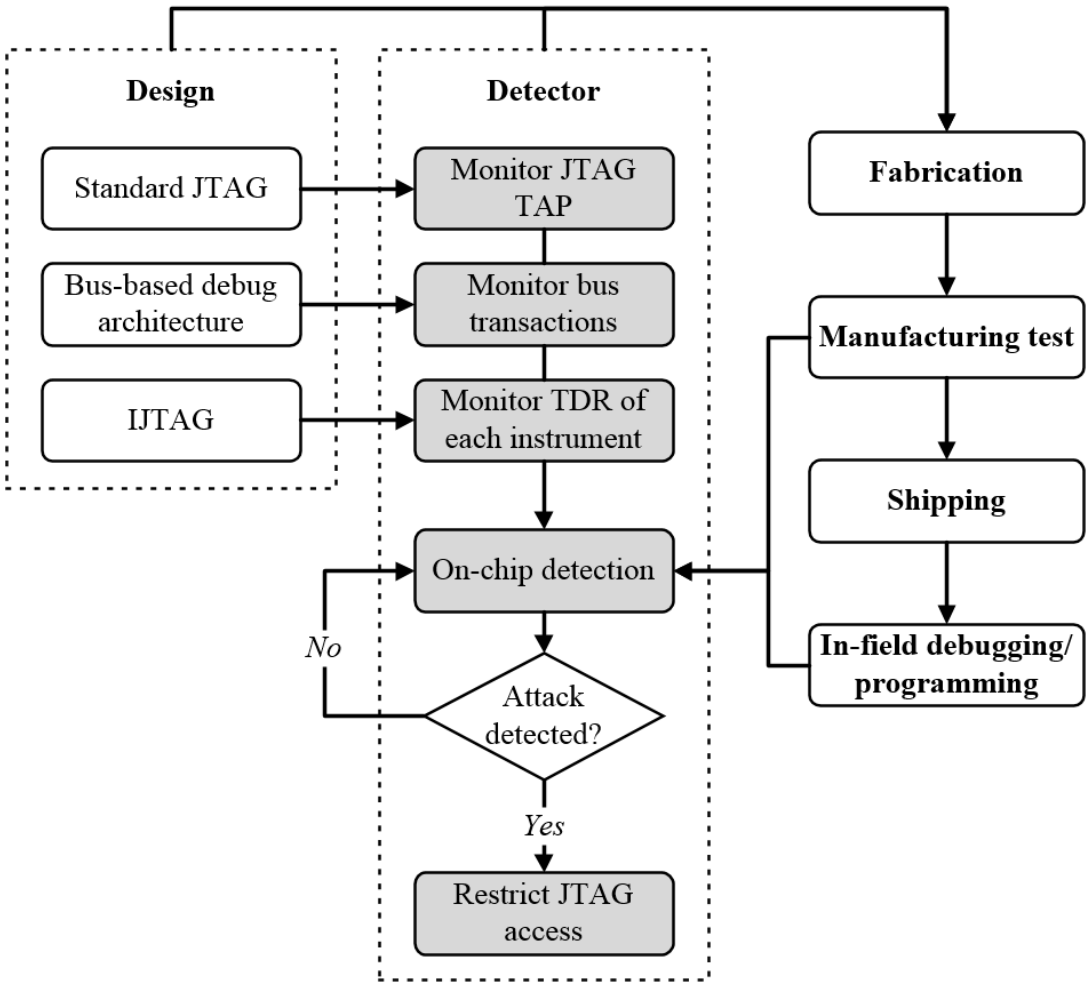


Figure 1.3: This diagram describes the methods of detecting and restricting illegitimate JTAG accesses developed in this dissertation.

less effective in detecting complex attacks, especially when the attacker already has basic knowledge of the JTAG. To detect more sophisticated attacks, one needs to study in depth how the JTAG is operated by legitimate users and attackers. A possible method involves comparing the monitored JTAG operation with a database of attack signatures [59] or legitimate traces [60, 61]. These deterministic methods, however, are less likely to tolerate the variance in JTAG operation, thus incurring false alerts. It will be demonstrated in a variety of experiments that JTAG operation is difficult to be described using deterministic rules. Thus, machine learning, because of its statistical nature, emerges as a promising technique for addressing these challenges.

Machine learning (ML) includes statistical models that equip a system with the capability of learning, i.e., progressively improving performance using the knowledge learned from previous data. Machine learning has shown a potential for addressing problems related to IC design, manufacturing, test, etc., especially as the complexity of modern ICs is increasing and significant amounts of data are produced [62]. When used for attack detection, an ML model is constructed based on legitimate JTAG operation and known attacks. This model is then used for monitoring JTAG operation and making online prediction of whether the user is legitimate or not. It is worth noting that the detection is stand-alone and the user can operate the JTAG directly without any prerequisite step. When compared to password-based encryptions whose major risk comes from leakage of the password, the detection methods developed in this dissertation minimizes this risk because the detector, already implemented on chip, does not need to be distributed to any user. When compared to challenge-response-based authentications, detection methods perform assessments locally without requiring the support of a network. Further, the methods developed in this dissertation can be used individually, or with prior work for achieving complementary protection of the JTAG.

Figure 1.3 shows an overall flow of detecting and restricting illegitimate access to the JTAG developed in this dissertation, with major contributions highlighted using dark boxes. Particularly, novel methods are developed to address how JTAG operation can be characterized, how ML models can be used for detecting potential attacks, and how to restrict access to the JTAG upon detection of an attack. When characterizing JTAG operation, three typical JTAG architectures are studied, namely a standard JTAG, a bus-based debug architecture, and an IJTAG network. In the remaining of this chapter more insights will be given for the developed methods.

In the rest of this chapter, more details will be elaborated upon for each part of the developed methods.

1.1 Characterizing JTAG/IJTAG Operation

Machine learning aims to construct a statistical model from data. The performance of a ML model depends on 1) the relevance of features, 2) the amount and quality of data, and 3) the effectiveness of the ML algorithm. Here, a *feature* is defined as an individual measurable property or characteristic of JTAG operation. Determining relevant features and collecting a large amount of high-quality data are prerequisites for constructing an effective ML model.

Features are outlined to characterize various aspects of JTAG operation. To determine effective features, we need to understand the architecture and basic operations of the JTAG. The IEEE Standard 1149.1 defines the test access ports and a boundary-scan architecture. The test access port (TAP) consists of four (or five) pins, namely the test data input (TDI), the test data output (TDO), the test mode select (TMS), the test clock (TCK), and an optional test reset (TRST). A boundary-scan chain, usually located at the periphery of a chip, not only allows test stimuli to be supplied serially to the chip via the TDI, but also allows the test response to be observed serially via the TDO. In addition to the boundary-scan chain, other registers, some through private test/debug functions, can also be accessed using the TDI and TDO ports. For example, some designs allow memory built-in self-test (MBIST) to be operated using the JTAG. To support the MBIST, the designer may implement registers that are used for configuring the mode of the MBIST, initiating the MBIST, and capturing the result of the MBIST.

Access to these data registers (DRs) is operated by the JTAG TAP controller. Every DR has one or more affiliated instructions that provide access, configuration, and/or control. A user can select a DR by loading the appropriate instruction opcode into the instruction register (IR) via the TDI. A selected DR can be operated using three fundamental register operations, namely capture, shift and update. When using the JTAG, a user typically operates the TAP controller for accessing the IR and then the selected DR. Therefore, JTAG operation can be characterized from three aspects, namely 1) the instruction opcode loaded into the IR, 2) the captured/loaded data in the DR, and 3) the control flow of the TAP controller. Moreover, feature engineering provides a means of

creating more features. Feature engineering aims at converting existing features into ones that better represent the underlying problem for the predictive model [63]. However, since there are no explicit rules for feature engineering, human effort is usually required.

After studying JTAG operation, a set of features are determined, but not all of them are relevant. Moreover, a feature may show different relevance for different JTAG architectures. This will be demonstrated using two representative JTAG architectures. For a standard JTAG architecture, a test/debug function is operated through executing one or multiple instructions in some specific order, so the sequence of instruction opcode is a very important aspect. For a system where debug interfaces (e.g., the JTAG) are connected to a bus, the on-chip components (e.g., processor registers and memory) are also accessed through the bus. In this scenario, the sequence of bus read and write exemplifies higher importance. Hence, the relevance of features needs to be evaluated using data collected from operating the JTAG. To collect a large amount of high-quality data, both legitimate and illegitimate scenarios of JTAG use should be studied comprehensively. In this dissertation, several open-source designs, including the OpenSPARC T1 [64] and T2 [30] processors, and the LEON3 processor [28], are used for experiments. For each design, the JTAG functions are studied, with traces of legitimate JTAG operation created, and attack traces, based on existing attacks to the JTAG, are also created.

This dissertation also studies the operation of an IJTAG network. Different from the standard JTAG, the IJTAG employs a reconfiguration scan network, where the scan chains within each instrument can be accessed and configured individually. Debugging an IJTAG network is operated by including/excluding the scan chains within each instrument and asserting/de-asserting the bits within the scan chains. Therefore, the bits, which control either the configuration of scan chains or the debugging/programming functions, are more relevant for characterizing the operation of an IJTAG network.

Note that there might exist other JTAG architectures whose operation can be characterized using different features. However, the JTAG designs considered in this dissertation are representative, providing guidelines for determining features and collecting data in other JTAG designs.

1.2 Detecting JTAG/IJTAG Attacks

Detection of JTAG attacks can employ deterministic approaches. For instance, an approach involves comparing the monitored trace with a database of pre-collected attack signatures, with an alert raised in case of a match [59]. This approach is commonly used in anti-virus software. Alternatively, the monitored trace can be compared with pre-defined state machines that describe legitimate JTAG operation, such that any operation deviating from the state machine is labeled as an attack [60, 61]. These deterministic approaches, however, may cause excessive false alerts due to the variance existing in both legitimate JTAG operation and attacks.

Machine learning is a preferable technique because it provides a statistical model and therefore can accurately model the variance in JTAG operation. Two common tasks of machine learning include supervised and unsupervised learning, as illustrated in Figure 1.4. In a supervised-learning task, data appear as pairs of an observation and a label. The label can be either a category that refers to the membership of the observation (the corresponding task is called *classification*), or a real number (the corresponding task is called *regression*) [65]. A supervised-learning task aims to construct a model that maps an observation to its most probable label, based on the collected data.

The process of constructing the model is called training, and the data used for constructing the model are called training data. Detection of JTAG attacks, to be addressed in this dissertation, is a supervised-learning and classification task because it aims to classify a trace of JTAG operation as either legitimate or attack.

After an ML model is trained, another data set is usually used for evaluating the performance of the trained model, and for avoiding the model overfitting the training data. This evaluation set is

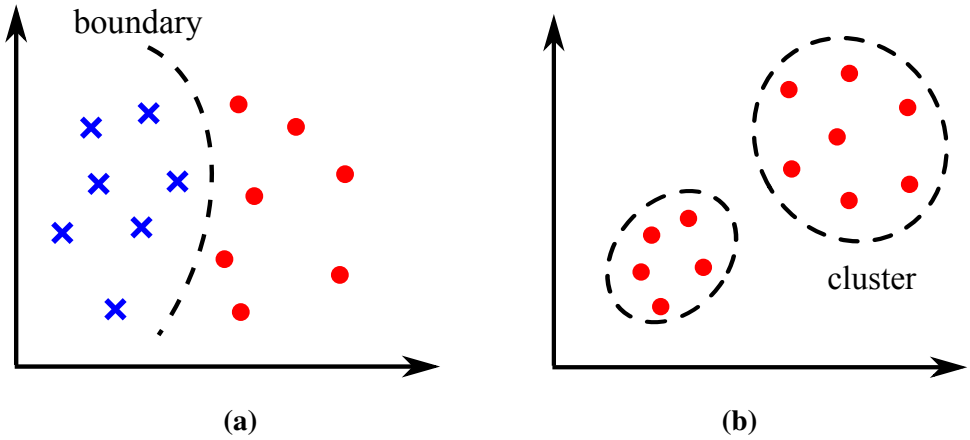


Figure 1.4: Supervised and unsupervised learning are illustrated using a classification and a clustering problem, respectively. (a) The goal of classification is to identify an optimal boundary between two or more classes, while (b) the goal of clustering is to group observations based on their density.

named a testing set. Many ML algorithms, such as decision tree [65], random forest [66], support vector machines (SVM) [67], artificial neural network (ANN) [68], and k-nearest-neighbor (k-NN) [65], can be used for constructing a classification model. In addition, sequential models, such as hidden Markov models [69] and recurrent neural network [70], commonly used for analyzing sequences, can also classify traces of JTAG operation. Distinguishing attacks from legitimate operation is not a trivial task. The data collected from benchmark designs demonstrate that the boundaries between legitimate operation and attacks are not easy to identify, and even worse, the overlapping between legitimate operation and attacks makes the classification even harder.

In an unsupervised-learning task, observations are not labeled. The objective, instead, is to discover hidden structures behind the observation. A central case of unsupervised learning is density estimation. A dense cluster of observations, if distinguished from sparse clusters, can be used as a criterion for anomaly detection. Particularly, an observation, if residing outside dense clusters, is more likely anomalous. In the scenario of JTAG attack detection, an anomaly detector, or named a one-class model, can be constructed based solely on traces of legitimate JTAG operation. When compared with binary models, a one-class model is typically less effective in identifying the classification boundaries [71], but more likely to detect unseen attacks [72].

To monitor real-time JTAG operation and disable access to the JTAG upon detection of an attack, the ML-based detector should be implemented on chip using either hardware or software. For a hardware implementation, the ML algorithm may pose a challenge in terms of overhead. If data collection and classifier training are processed online, then a large amount of data need to be stored on chip and complex logic is required for supporting the training process. To avoid this huge overhead, the classifier, developed in this dissertation, is trained offline, with only the trained classifier stored on chip. Even so, the classifier may still consume a large amount of chip area, especially if the classifier is trained using high-dimensional data. For example, the data collected from an IJTAG network can have a high dimensionality, considering that each instrument, including both their configuration status and the test data supplied to them, needs to be monitored. This high-dimensional data usually lead to a classifier with large size. To make on-chip detection possible, an on-chip data compression method is developed to reduce the dimensionality.

Besides the performance of attack detection and hardware implementation, other issues, such as software implementation of the detector, detection of unknown JTAG attacks and attacks to the ML classifier, will also be studied in this dissertation.

1.3 Restricting JTAG Access

Access to the JTAG should be restricted upon detection of an attack. The objective of access restriction is to prevent on-chip data from being observed and to prevent on-chip system from being controlled. A method adopted in industry is to disable access to scan chains completely [50] or partially [62, 73]. A complete disabling prevents in-field access to the JTAG through an anti-fuse, while a partial disabling only excludes flip-flops that contain sensitive information (e.g., cryptographic keys) from the scan chain. Some ICs can be programmed, through eFuse for instance, to allow multi-level accesses. An industrial example involves the i.MX6 series processors developed by Freescale that define three security modes for the JTAG, namely, disabled, boundary-scan-only, and enabled. Advanced DFT techniques, including on-chip compression [74], X-tolerance [13],

and X-masking [75,76], also restrict observability provided by the JTAG, but bypassing these DFT infrastructures is usually possible for the purposes of diagnosis and in-field debugging. Another method, described in [77], aims to avoid leakage of sensitive data caused by switching from the normal mode to the test mode. Particularly, a switch from the normal mode to the test mode will reset the flip-flops containing sensitive information [15]. Obfuscating test output is another effective method that can prevent observability. To obfuscate the output, the architecture of scan chains is modified through scrambling the order of sub-chains [78–80], or inserting multiplexors [55,81,82], inverters [83, 84], and dummy flip-flops [85] into scan chains.

All these methods can be employed for restricting JTAG access. However, in this dissertation, a new method is developed for securing the JTAG, which can improve the security of the attack detection described before. Specifically, the JTAG architecture is modified, such that access to the JTAG is restricted upon detection of an attack. The restriction not only prohibits observability and controllability provided by the JTAG, but also obfuscates the JTAG operation that triggers the restriction. This is important because if the attacker knows which JTAG operation triggers the restriction, he/she can then avoid that operation in future attacks. Knowing which operation is labeled as legitimate and which is labeled as an attack, an attacker can gain details of the detector; even worse, repeating this process may even leak all information of the detector. However, if the operation triggering the protection is obfuscated, then the attacker would obtain much less information about the detector, such that reverse engineering the details of the detector becomes much more difficult.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows. In Chapter 2, the standard JTAG and the bus-based debug architecture are studied, with JTAG operation characterized using relevant features. More, typical strategies of attacking the JTAG are summarized according to existing work. Chapter 3 describes how ML models are employed for distinguishing JTAG attacks from legitimate

operation. The performance is evaluated using several open-source designs. Chapter 4 discusses attack detection for an IJTAG network. To fit high-dimensional data for on-chip detection, a novel method of compression is developed for reducing the dimensionality of collected data. In Chapter 5, hardware overhead is evaluated for the detector, and a secure JTAG architecture is developed to prevent observability and controllability provided by the original JTAG. Finally, Chapter 6 concludes the dissertation and discusses future works.

Chapter 2

Characterization of JTAG Operation

Despite the variety of possible architectures, JTAG designs share similar goals related to test and debug, and therefore share similar ideas and philosophy of design for test/debug functionality. This also makes it possible to study and even generalize the operation of JTAG. This chapter will elaborate upon feature characterization and data collection for JTAG operation, which act as prerequisites for constructing an effective ML model. First, the JTAG of various systems, including the OpenSPARC T1, T2, and the LEON3 processors, are studied in Section 2.1. Then, the attacks of uncovering the architecture of the JTAG are described in Section 2.2. Section 2.3 presents four generic categories of features for characterizing JTAG operation, and then proposes specific features for each category. Finally, in Section 2.4, both legitimate JTAG operation and attacks are created for the benchmark designs on which the importance of each feature is evaluated.

2.1 JTAG Architecture

After fabrication, a chip needs to be tested for checking if it functions as expected, free of defects. JTAG, defined by the IEEE Standard 1149.1, was designed to assist with device, board, and system testing, diagnosis, and fault isolation. Today, JTAG is used as the primary means of

accessing subsystems of ICs. This section describes the standard JTAG architecture and a debug architecture where the debug commands/data are communicated through a bus.

2.1.1 Standard JTAG Architecture

The IEEE Standard 1149.1 defines the test access port and a boundary-scan architecture. The test access port (TAP) is an interface added to a chip, consisting of four (or five) pins, including the test data input (TDI), the test data output (TDO), the test mode select (TMS), the test clock (TCK), and the optional test reset (TRST), as shown in Figure 2.1. A boundary-scan architecture, usually connecting internal pins as a daisy chain and located at the periphery of a chip, allows serial access to internal pins via the TDI and the TDO ports. When multiple chips are integrated on a board, the TAP interface allows each chip to be daisy-chained, such that a test probe only needs to connect to a single JTAG port for accessing all chips on the board. The boundary-scan chain not only allows test stimuli to be supplied serially to the chip pins via the TDI, but also allows the test response to be observed serially via the TDO. Other than a boundary-scan chain, 1149.1 also defines a mandatory bypass register (a one-bit flip-flop) and an optional ID register (storing data with a standardized format, including manufacturer code, a part number assigned by the manufacturer, and a part version code). Each of these registers is defined with an opcode, such that the register can be selected (i.e., connected between the TDI and TDO ports) if its corresponding opcode is loaded into the instruction register.

The JTAG standard allows implementation of additional data registers (DRs), in order to support testing, diagnosis, and in-field debugging. The design-for-testability (DFT) designer has flexibility to decide which test/debug functions to implement, which DRs to add for supporting the functions, which JTAG instructions to define for operating the DRs, and how to encode the instructions. The added DRs are usually used by authorized users (e.g., the manufacturer and the test engineer) and not revealed to end-users. An example involves the internal scan chains that exist in most chips. Particularly, flip-flops within a chip are typically connected as one or multiple scan

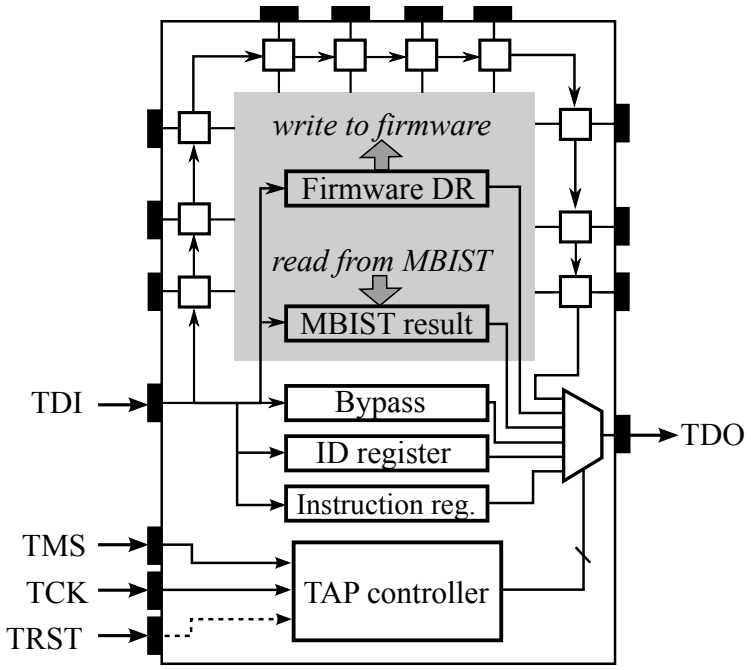


Figure 2.1: The block diagram of a typical JTAG architecture.

chains when the chip is switched from the normal mode to the test mode. These scan chains can be accessed via the TDI and TDO ports, equipping the IC with high-level testability. A user, if aware of how to access these scan chains, can not only observe the values stored in every flip-flop of the scan chains, but also load data to specific segments of the scan chains. Besides scan chains, other DRs, some through private JTAG functions, can also be accessed using the TDI and TDO ports. Figure 2.1 shows two example DRs, one for updating the firmware, and the other for capturing the result of the memory built-in self-test (MBIST).

Access to DRs is operated by the JTAG TAP controller that implements the state diagram shown in Figure 2.2. The JTAG TAP controller is a finite state machine (FSM) controlled by the TMS port. Every DR has one or more affiliated instructions that provide access, configuration, or control. A user can select a DR by loading the appropriate instruction opcode into the instruction register (IR) via the TDI pin. A selected DR can be operated using three fundamental register operations, namely capture, shift, and update. For example, for ICs, selecting the boundary-scan chain initially requires an all-zero opcode. The user then loads the input stimuli via the TDI by

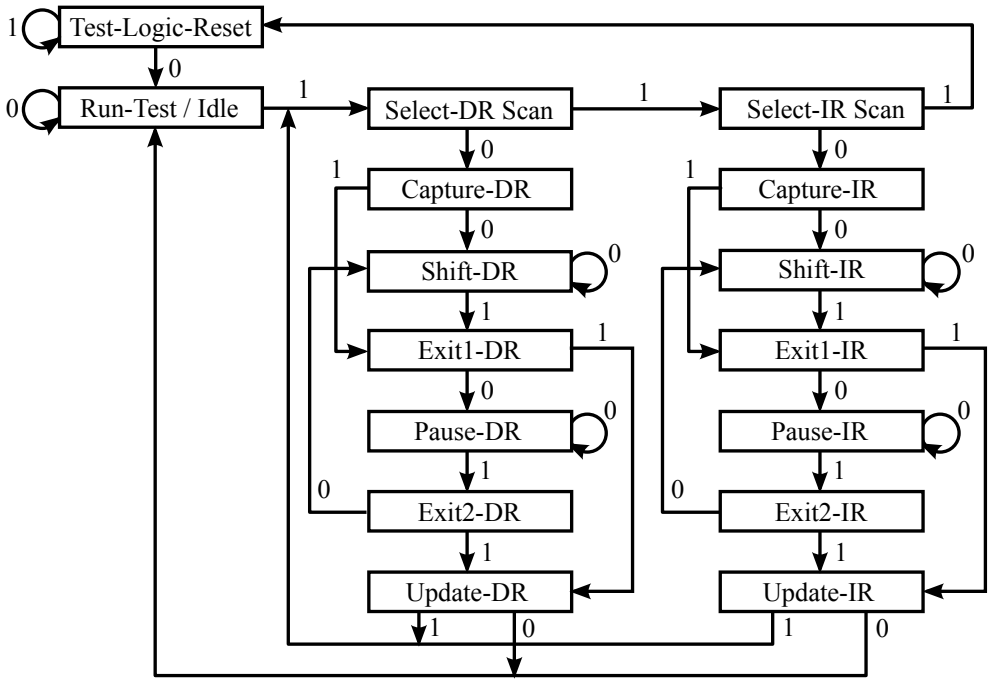


Figure 2.2: The JTAG TAP controller is a one-input FSM.

repeatedly entering the SHIFT-DR state. Finally, the UPDATE-DR state applies the stimuli as input to the chip logic. After the operation of the chip logic is completed, the response is captured by the scan chain within the CAPTURE-DR state, which can then be shifted out via the TDO by repeatedly entering the SHIFT-DR state. Another example involves the MBIST operation. Different from the boundary scan, the MBIST operation of the OpenSPARC T2 is executed using a sequence of instructions with a predefined order, that is, typically, selecting the on-chip modules to be tested (instruction MBIST_BYPASS), setting the MBIST mode (MBIST_MODE), initiating the MBIST process (MBIST_START), and checking the result (MBIST_RESULT).

2.1.2 Bus-based Debug Architecture

Many system-on-a-chip (SoC) designs employ a bus architecture, meaning that the functional blocks in a SoC are interconnected using a bus. Some bus examples involve the AMBA AHB developed by ARM [86] and the WISHBONE developed in OpenCores [87]. The ARM Advanced

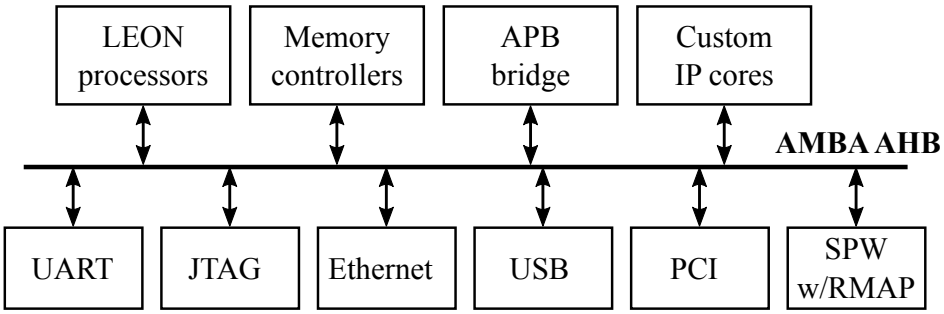


Figure 2.3: The blocks of the LEON3 are connected through an AMBA Advanced High-performance Bus (AHB) [3].

Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect specification. Because the AMBA facilitates development of multi-processors with a large number of controllers and peripherals, it is widely used in ASICs and SoCs, including application processors used in modern portable mobile devices. The AMBA defines an Advanced High-performance Bus (AHB) protocol. The AHB access to a target block is controlled through a non-tristate multiplexor, thereby admitting bus access to one bus-master at a time. The LEON3, a 32-bit processor compliant with the SPARC V8 RISC architecture, is an example that employs the AMBA AHB protocol [28]. As shown in Figure 2.3, the blocks of the LEON3, including the processors, the memory controller, the custom IP cores, and the interface ports, are connected through an AMBA AHB bus [3].

The LEON3 has multiple debug interfaces, such as UART, JTAG and Ethernet. The LEON3 processor implements a debug mode, during which the pipeline is idle and the processor is controlled by a specified debug master. More precisely, a debug support unit (DSU) is used as the AHB slave through which, the AHB master, including the debug interfaces, can access the processor during the debug mode. Figure 2.4 shows that the JTAG TAP is used for accessing the processors through the DSU [3]. The DSU can be accessed at any time, while the processor registers can only be accessed when the processor is placed in the debug mode.

A transaction on the AHB consists of an address phase and a subsequent data phase. More precisely, an AHB master sends an address that specifies both the target AHB slave and the location of the target register, and then, after two bus cycles, sends data into the target register or captures

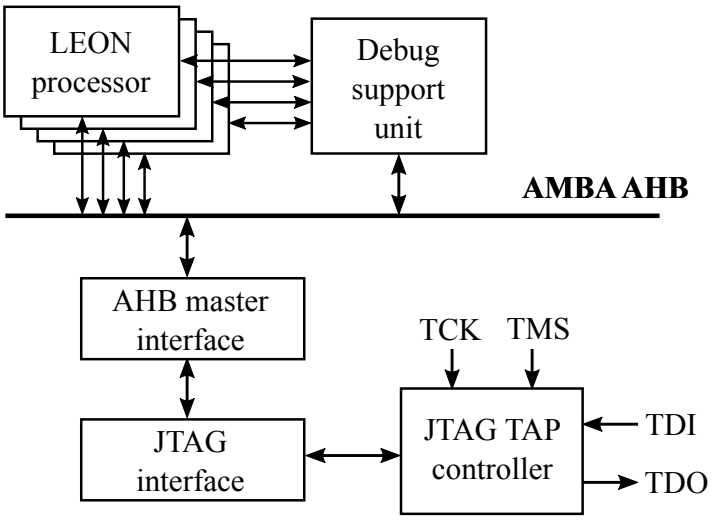


Figure 2.4: The JTAG, acting as an AHB master, can access processor registers through a debug support unit (DSU) [3].

data from the target register. When the JTAG is used as the debug interface, two data registers, namely an address register and a data register, are implemented to comply with the AMBA AHB protocol. To perform a read access, the user needs to shift a command, consisting of a read bit, the AHB access size, and the AHB address, into the address register. Then, an UPDATE-DR state initiates the read access, and the data is ready to be shifted out from the data register. A write access is performed by shifting the command, consisting of a write bit, the AHB size and the AHB address, into the address register, followed by shifting the write data into the data register. Subsequently, an UPDATE-DR state initiates the write access.

Through the bus, a debug host can access processor registers (e.g., the instruction trace buffer, the program counter register, and debug control register), caches, and memory. A debug host can also control execution of the processors, for example, by inserting break points into a program. These features are critical for software debugging. However, if not secured, the debug interfaces might be misused deliberately to undermine the security of software. It is also worth noting that not only the JTAG, but also other debug interfaces need to be monitored, because all of them can be used for controlling the bus. In this case, it might be more appropriate to monitor bus transactions rather than JTAG operation.

TABLE 2.1: A LIST OF VARIOUS NON-INTRUSIVE ATTACKS.

Non-intrusive attack	Target / Objective	Details
Differential attack [8–14, 75, 88–93]	Steal cryptographic key from scan chains	An attacker inputs challenge pairs, running the cryptographic algorithm, and compares the outputs
Test-mode-only attack [15, 74]	Steal cryptographic key	Instead of switching between normal mode and test mode, the key may be stolen in only test mode
Replay attack	Sniff password	If all chip instances share the same password (even hashed), then an attacker can eavesdrop the password
Resetting and flushing attack [83, 84]	Reset or identify key bits	For scan chains obfuscated by inverted bits and modified bits, resetting or flushing the scan chains can reveal the modified bits
Mass erase of eFuse or flash memory [43–45]	Reset the key	Some industrial designs use bits in eFuse or flash memory for restricting JTAG access, while a mass erase can reset the bits
Fault injection attack [94, 95]	Modify memory bit dynamically	An Android system defines bits controlling privilege of accessing resources, which can be modified using the JTAG
Logic analysis [96]	Uncover combinational logic	Scan chains unfold sequential logic into combinational logic that can be reverse engineered
Reverse engineering [36, 57]	Uncover JTAG functions	An attacker investigates how to use the JTAG to operate the hidden functions

2.2 JTAG Attacks

Various non-intrusive attacks via the JTAG have been reported, as summarized in Table 2.1. Existing attacks to cryptographic keys assume that the intermediate state of a multi-stage encryption is stored within one or more scan chains. An attacker can run the encryption for one stage, and then switch the chip to the test mode such that intermediate results of the encryption can be observed

through the scan chains. Then the key can be derived using differential attacks, i.e., inputting challenge pairs, running the cryptographic algorithm, and comparing the outputs [8–14, 75, 88–93]. The attack in [97] shows that the secret key in DES can be retrieved successfully using less than 30 plain-texts on average and 32 plain-texts in the worst case. Further, attacks may still be possible even with the existence of countermeasures. One example involves test-mode-only attacks that can derive the key even if the system is test-mode-protected (e.g., sensitive data is erased once the chip is switched from the normal mode to the test mode) [15, 74]. Replay attacks occur when chip instances share the same password. Particularly, if the password used for authentication is sniffed by an attacker, then the attacker can reuse the password to access the chip. Resetting/flushing attacks are used when scan chains are modified by inserting inverters or multiplexors between scan flip-flops. The work in [83, 84] demonstrates that resetting or flushing the scan chains can reveal the modified bits. In some real-world products, bits for restricting JTAG access are stored in eFuse or firmware [43–45]. However, these bits, if too simple, can be reset using a mass erase. The work in [94, 95] shows that fault injection attacks can also be conducted using the JTAG. Particularly, an attacker can successfully modify a control bit in the memory of the Android OS, such that a higher privilege of accessing system resources can be gained. Another possible attack involves logic analysis [96]. Since the test mode essentially converts sequential logic into combinational logic, an attacker may use scan chains to uncover the logic.

To conduct an effective attack, an attacker needs to know how to operate the JTAG functions, including accessing scan chains. The JTAG of a processor design may implement dozens of JTAG instructions and data registers [30, 64]. The way of accessing on-chip components, such as scan chains, system registers, on-chip RAM, eFuse, and non-volatile memory, is commonly not revealed to end users. Thus, reverse engineering the unrevealed (or private) JTAG functions serves as a prerequisite step for further attacks. Table 2.2 lists typical attack strategies, which can be categorized as three main steps and nine sub-steps [36, 57]. The first step is to identify the JTAG ports physically. This step is relatively easy because the test ports are usually clustered together someplace on the chip. The second step is to explore the property of the IR and the DRs. The length of the

TABLE 2.2: JTAG ATTACK STRATEGIES ARE ORGANIZED IN THREE MAIN STEPS AND NINE SUB-STEPS.

Steps		Sub-steps
Locate test ports	S_1	Identify the location of the JTAG ports on chip/board
Find DR property	S_2	Identify the length of the IR
	S_3	Identify the length of each DR
	S_4	Find the opcodes that do not correspond to any DR
	S_5	Check if each DR can be captured/updated
	S_6	Identify internal scan chains
Interrogate private functions	S_7	Separate opcodes into bundles based on their associated functions
	S_8	Determine the nature (control vs. data) of each DR
	S_9	Investigate adjacent instruction opcodes for characterizing any interactions between them

IR and the DRs can be easily extracted by shifting in a sufficiently long sequence of ones followed by a single zero. Undefined opcodes, i.e., not associated with any DR, usually behave as a one-bit bypass register or a disconnection. These opcodes are important because they likely lie between valid opcodes corresponding to different JTAG functions. For example, in the OpenSPARC T2, the instructions for control-register configuration and MBIST have opcodes that are separated by undefined opcodes 11 and 12 (in hexadecimal format). Besides checking the length of a DR, it is also beneficial to learn if a DR can be captured and updated. An attacker may dump on-chip data from a DR that can be captured, and write data into the chip through a DR that can be updated. Further, the attacker can employ a trial-and-error method to investigate the JTAG functionalities controlled by the instructions and the DRs. For example, if a DR can be updated and another DR with neighboring opcode can be captured, then an attacker can make a reasonable guess that these two DRs are used for reading some memory. In other words, the first DR sends an address and the second DR captures the corresponding data. Using similar means, the attacker can uncover more JTAG functions by investigating the interactions between adjacent instruction opcodes.

2.3 Feature Characterization

To better characterize JTAG operation, we define an operation cycle based upon using the JTAG. An *operation cycle* starts from the RUN-TEST/IDLE state, loads an instruction opcode into the IR by traversing the states in the right column of the FSM shown in Figure 2.2, then captures/updates data from/to the selected DR by traversing the states in the middle column, and finally ends in the RUN-TEST/IDLE state. With the definition of an operation cycle, JTAG operation can be regarded as a sequence of operation cycles. During each operation cycle, four categories of features, as shown in Equation 2.1, will be constructed. The first two categories (F_{ir} and F_{dr}) characterize the opcode loaded into the IR and the data loaded into the selected DR, respectively. The third category (F_{fsm}) characterizes state traversal of the FSM shown in Figure 2.2. The fourth category (F_{ext}) includes features that require information outside the operation cycle. The difference between F_{ir}, F_{dr} and F_{fsm} can also be distinguished by the fact that F_{ir} and F_{dr} are based on data provided via the TDI port, while F_{fsm} is based on the bit stream provided to the TMS port.

$$Features = \left(F_{ir}, F_{dr}, F_{fsm}, F_{ext}\right) \tag{2.1}$$

Equation 2.1 gives a guideline for constructing features, but does not provide specific features. Constructing specific features, however, relies on the underlying problem. Table 2.3 shows some features used in this work. These features are considered because they reflect some aspects of JTAG operation, and therefore aid in the detection of JTAG attacks. For example, the feature TMS_TRANSIT, representing the number of TMS transitions (either from zero to one or from one to zero), is relevant in two circumstances: 1) reset-type instructions require fewer TMS transitions because they do not select any DR, and 2) entering the PAUSE-DR state, which is necessary if shifting a DR needs to be temporarily-suspended, leading to more TMS transitions.

Some features, representing deterministic rules, can be used as sanity checks for JTAG operation. For example, when a DR is selected, the number of shifting cycles should be equal to

TABLE 2.3: SOME FEATURES USED FOR CHARACTERIZING JTAG OPERATION.

Category	Feature	Description	Format
F_{ir}	OPCODE	The opcode loaded into the IR during the UPDATE-IR state	Integer
	OPCODE_CHANGED	Whether the opcode is changed compared to the previous operation cycle	Binary
F_{dr}	BUS_ADDR	The data loaded into the address register during the UPDATE-DR state	Integer
	BUS_DATA	The data loaded into the data register during the UPDATE-DR state	Integer
	BUS_RW	Whether the bus operation is read or write	Binary
F_{fsm}	CYC_SHIFT_DR	No. of clock cycles within the SHIFT-DR state	Integer
	CYC_SHIFT_DR_EQ	Whether no. of clock cycles within the SHIFT-DR state is equal to the length of the selected DR	Binary
	CYC_TLR	No. of clock cycles within the TEST-LOGIC-RESET state	Integer
	TMS_TRANSIT	No. of TMS transitions	Integer
F_{ext}	LEGAL_OPCODE	Whether the opcode corresponds to a valid JTAG function	Binary
	TYPC_TRANSIT	Whether the transition from one instruction to the next is typical	Binary
	PREV_PRED	Detection result of the previous operation cycle	Binary

the length of the DR, for either a read or write operation (this is described by the feature CYC_-SHIFT_DR_EQUAL). However, this rule might be violated if an attacker does not know the length of the DR. Another example involves the feature LEGAL_OPCODE that describes whether the opcode corresponds to a valid JTAG function. An n-bit IR allows 2^n opcodes, meaning that at most 2^n instructions can be defined. However, typically only a subset of the possible opcodes are used for achieving specific functions, while the others remain undefined. The undefined opcodes may select the BYPASS register, or simply behave as a disconnection, depending on the design. Typically, undefined opcodes seldom have a default functionality besides the BYPASS operation. If an undefined opcode is loaded into the IR, then the user might be an attacker.

The features, listed in Table 2.3, may show different levels of importance for different JTAG designs. For example, for a standard JTAG architecture, OPCODE might be more important because it indicates the DR and the on-chip component that the user aims to access, while for a SoC design that employs a bus-based debug architecture, the bus-related features, such as BUS_ADDR, BUS_-DATA, and BUS_RW, are more relevant. Therefore, the choice of features relies on the underlying JTAG design. For a new JTAG, one might need to construct additional features to better capture the characteristic of its operation.

Even though more features are preferable for ML classification, coming up with new features is typically difficult, time-consuming, and requires expert knowledge [98]. More, not only the quantity but also the quality of features impacts the performance of ML models. Feature engineering provides a means of converting raw data into features that better represent the underlying problem for ML, resulting in improved model accuracy on unseen data [63]. However, the search of high-quality features also relies on the underlying problem, meaning that there are no universal rules. One example of feature engineering involves some features of the category F_{ext} as shown in Table 2.3. Specifically, the feature PREV_PRED represents the previous prediction made by the ML model, while the feature TYPC_TRANSIT, requiring a pre-constructed look-up table that records frequent opcode transitions, indicates if the observed transition of opcode is typical. Other possible features (not shown in Table 2.3) include temporal derivatives (i.e., the value change for some feature during two successive operation cycles) and basic statistics (e.g., the maximal, minimal, and mean values for some feature within a series of operation cycles).

Not all features constructed through brainstorming and feature engineering are relevant for model performance, so a further step of feature selection is necessary. Moreover, selecting only relevant features can also reduce dimensionality of the data, thus being beneficial for reducing the overhead of online detection. There exist many methods for feature selection, such as removing features with low variance, computing correlation between features and the class label [99], recursive feature elimination [100], and tree-based feature selection [101]. Some of these methods will be used for evaluating importance of the extracted features.

2.4 Experiments

This section describes feature characterization and data collection for several open-source designs, namely, the OpenSPARC T1 and T2 processors developed by Oracle [30, 64], and the LEON3 processor developed by Gaisler [28]. For each design, both legitimate JTAG operation and attacks are created, which are used to identify effective features.

2.4.1 OpenSPARC T1 and T2

Both the OpenSPARC T1 and T2 are a multi-core, 64-bit multiprocessor, with a well-designed JTAG. As shown in Table 2.4, the OpenSPARC T1 implements various JTAG functions for test and debug, while the OpenSPARC T2 adds more functions for facilitating more powerful access to on-chip components.

Different from the OpenSPARC T2, the JTAG of the OpenSPARC T1 employs a composite IR. Particularly, a part of the IR (six bits) is used for the instruction opcode, while the remaining portion (twelve bits) is used for configuration. For instance, when operating the clock, the user first needs to load the opcode corresponding to the clock operation into the first six bits, and then use the remaining twelve bits for specifying the targeted component (i.e., CPU, DRAM, or J-bus). The OpenSPARC T2, instead, implements these configuration bits as DRs, and correspondingly, defines additional opcodes for selecting the DRs. This also explains why the OpenSPARC T2 has more DRs.

For each JTAG function listed in Table 2.4, a set of legitimate operations are created, as summarized in Table 2.5. The operations are represented as *traces*, each consisting of a sequence of JTAG instructions. Each trace aims to achieve some specific operation, such as initiating the MBIST and updating specific bits of the eFuse. In addition to legitimate JTAG operation, JTAG attacks are created based on the attack strategies described in Section 2.2.

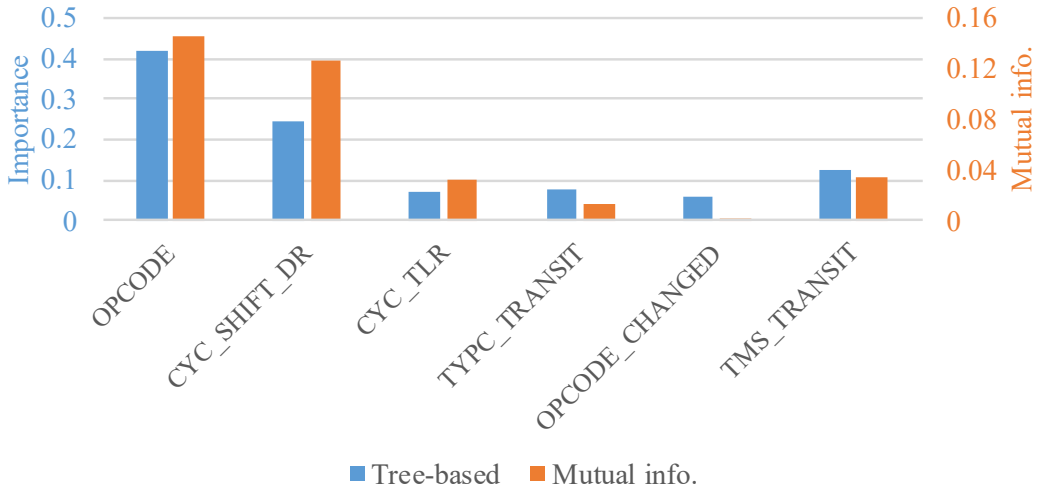
TABLE 2.4: THE JTAG FUNCTIONS IMPLEMENTED IN THE OPENSPARC T1 AND T2.

JTAG function	OpenSPARC T1		OpenSPARC T2	
	No. of DRs	No. of inst.	No. of DRs	No. of inst.
Public	3	6	3	8
Clock control	0	3	4	7
Control register	4	9	3	8
Memory BIST	1	4	5	9
Direct memory observation	-	-	2	3
Electronic fuse	5	7	6	8
Shadow scan	4	4	9	9
Internal scan	20	7	34	5
Debug configuration register	-	-	8	19
L2 cache	-	-	3	4
Logic BIST	-	-	3	6
Total	**37**	**40**	**80**	**86**

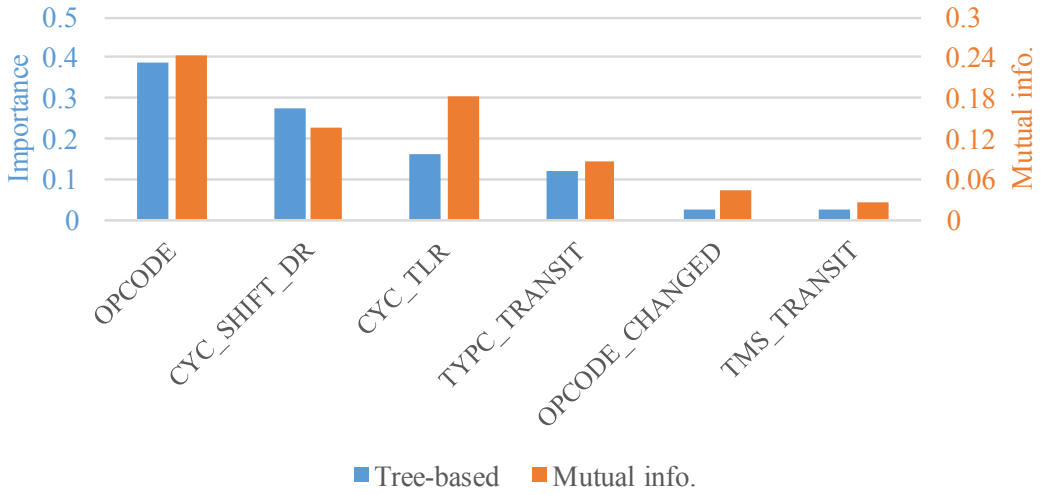
TABLE 2.5: LEGITIMATE JTAG OPERATION AND ATTACKS ARE CREATED FOR THE OPENSPARC T1 AND T2.

		OpenSPARC T1	OpenSPARC T2
Legitimate operation	No. of traces	280	814
	No. of inst.	24,765	136,417
Attack	No. of traces	343	4,096
	No. of inst.	20,062	169,042

After simulating legitimate and illegitimate (attack) traces on the OpenSPARC T1 and T2, the data, corresponding to the features listed in Table 2.3, are collected for every operation cycle. Note that some features (i.e., CYC_SHIFT_DR_EQ and LEGAL_OPCODE) are used as sanity checks, and thus will not be considered for building the ML model. The feature PREV_PRED requires the ML model to provide the earlier prediction result and will be considered in the next chapter. For the remaining features, their importance is evaluated using the tree-based feature selection, recursive feature elimination, and the mutual information between each feature and the class label. As exhibited in Figure 2.5, all of these methods indicate that OPCODE and CYC_SHIFT_DR are



(a)



(b)

Figure 2.5: The importance of features is calculated using a decision tree and mutual information for (a) the OpenSPARC T1 and (b) the OpenSPARC T2. In each chart, the results of recursive feature elimination is indicated by the order of the features, i.e., features on the right will be eliminated firstly.

important features. Figure 2.5 also shows that, for the OpenSPARC T2, the sequential feature (reflected by TYPC_TRANSIT) is quite important for characterizing JTAG operation.

An operation cycle is now represented by a h-dimensional vector, and a trace is then represented by a sequence of h-dimensional vectors. To better capture the sequential characteristic of JTAG operation, we use a sliding window (with four operation cycles) to reorganize the data, that is, concatenating every successive four vectors as larger vector. The reorganized data are plotted

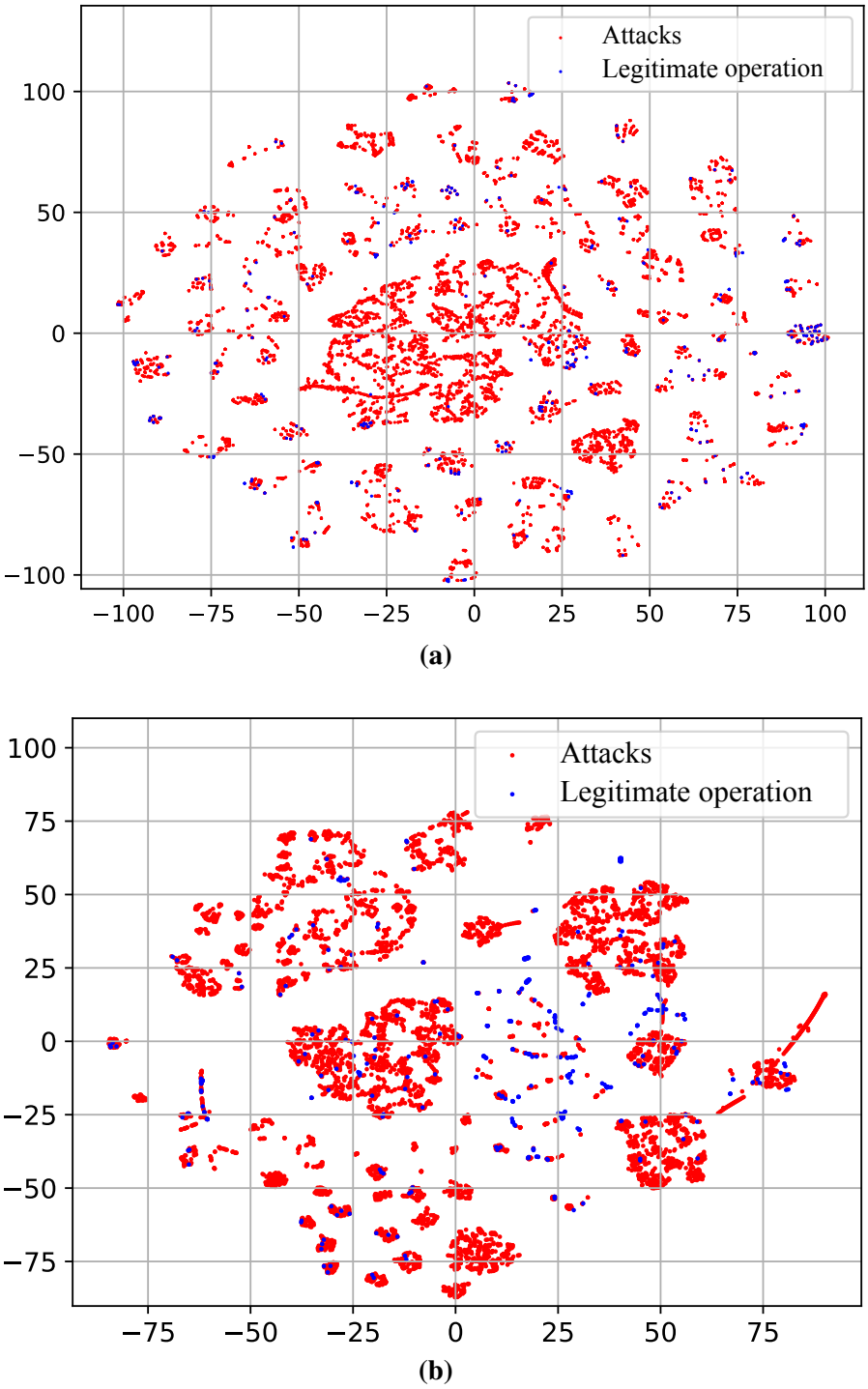


Figure 2.6: Legitimate JTAG operation and attacks for (a) the OpenSPARC T1 and (b) the OpenSPARC T2, are reorganized using a sliding window, and then plotted using t-SNE.

using the technique of t-SNE[1] [102], as shown in Figure 2.6. Figure 2.6 demonstrates that attacks

[1]The t-SNE employs a non-linear method that converts high-dimensional data into two or three dimensions, while maintaining the similarity for neighboring data samples.

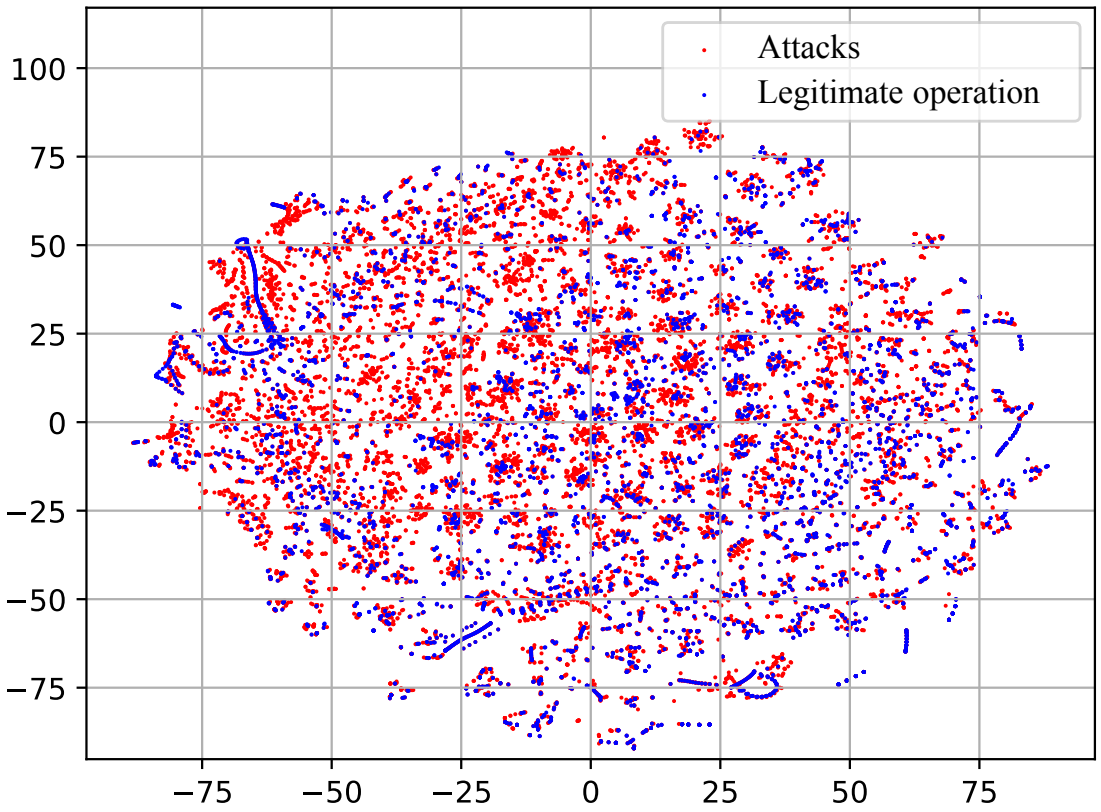


Figure 2.7: Legitimate JTAG operation and attacks via the bus of the LEON3 processor are reorganized using a sliding window, and then plotted using t-SNE.

and legitimate JTAG operations not only share a nonlinear boundary but also have overlapping regions.

2.4.2 LEON3

The LEON3 implements two data registers, namely an address register and a data register, within the JTAG, in order to support bus access. These two registers can be operated by a debug tool called GRMON [103], provided by Gaisler. A user can program in the GRMON, which is then compiled into bus transactions. The GRMON supports various debug functions, such as read/write access to system registers and memory, breakpoint management, disassembler and trace buffer management.

Through studying the documentation of the GRMON, a set of legitimate debug traces are created, which are then converted into sequences of bus read and write operations. Attack traces are also created based on the strategies described in Section 2.2. The assumptions made for the attacker still hold, that is, the attacker does not know which debug functions are implemented, and how to access the bus for operating these debug functions.

The JTAG of the LEON3 has significantly fewer instructions than the OpenSPARC T1 and T2, because most of the debug functions are achieved through bus operation. As a result, because only the bus-related registers are involved in debugging traces, we only consider BUS_ADDR, BUS_-DATA and BUS_RW. Both the address and the data have 32 bits, which however can represent an integer up to $2^{32} - 1$. To avoid possible overflow during the following classification, the 32 bits are divided into four bytes. Each bus transaction is then represented as a nine-dimensional vector, i.e., four for the address, four for the data, and the other one indicating whether the transaction is a read or a write.

Like the OpenSPARC T1 and T2, the sequences of bus transactions are reorganized by a sliding window, that is, every eight successive bus transactions are concatenated. To better understand how well the data are separated, they are plotted in Figure 2.7. Figure 2.7 clearly demonstrates that legitimate debug operations are very similar to attacks, which challenges attack detection.

2.5 Summary

This chapter discusses feature characterization and data collection for JTAG operation, both of which are significant for constructing an effective ML model for detecting JTAG attacks. More precisely, JTAG operation is characterized using a set of features, whose importance is evaluated using a tree-based feature selection method. The importance of a feature may vary for different designs, and thus it is necessary to determine relevant features for a specific design. This is verified using three benchmark designs, namely the OpenSPARC T1, T2, and the LEON3 processors. For

each design, the test/debug functions supported by the JTAG are studied comprehensively. A set of legitimate JTAG operation created, and attacks to the JTAG, summarized based on prior work, are also created. With features determined and data collected, the next step, i.e., building the detection model, will be discussed in Chapter 3.

Chapter 3

Detection of JTAG Attacks

As described in Chapter 2, JTAG operation can be viewed as a sequence of operation cycles, and correspondingly, the data collected from JTAG operation consists of multi-dimensional observations over time. This chapter will elaborate on how these data can be analyzed for detecting illegitimate access to the JTAG. Specifically, Section 3.1 describes a sliding window for converting sequences into fixed-size vectors, such that a variety of ML algorithms, described in Section 3.2, can be applied for classification. The accuracy of sliding-window methods can be improved using a delayed labeling method, which is described in Section 3.3. A fixed-size window, however, might still not characterize JTAG operation accurately. On one hand, an observation might be correlated with an observation that is outside the window, and on the other hand, the observations captured within a window might not correlate with each other. For this reason, sequence models, which can characterize serial dependence more accurately, are evaluated in Section 3.4. In addition, because new attacks are likely on the horizon, implying that the effectiveness of a detector strongly relies on its capability of detecting new, unseen attacks. To address this issue, a cascaded classifier is described in Section 3.5. In Section 3.6, accuracy of the detectors is evaluated using benchmark designs. Finally, Section 3.8 summarizes the chapter.

3.1 Detection Using a Sliding Window

JTAG operation can be viewed as a sequence of operation cycles, and correspondingly, the data collected from JTAG operation involve a series of multi-dimensional observations. Classifying the observations within an individual operation cycle does not capture serial dependence. In other words, observations within an operation cycle may depend on prior operation cycles. For identifying attacks accurately, both the current and prior operation cycles should be inspected. However, determining the number of operation cycles is not a trivial task because serial dependence varies in different cases. Some observations may only depend on recent observations, while others might correlate with even earlier ones. Sequence models, such as the long short-term memory (LSTM) networks [70], can learn serial dependence from data that involves variable amounts of historic information can be retained. However, using sequence models for online detection is challenging due to their complexity. Sequence models, typically running in software, cannot make instant predictions; however, if implemented using hardware, they incur significant overhead.

A fixed-size window serves as a trade-off between model complexity and capturing serial dependence. That is, a window of w successive operation cycles is considered for classification. In addition, the windows are collected in an overlapping manner (referred to as *sliding windows*), meaning that a prediction will be made upon completion of every operation cycle, based on the current cycle and the prior $(w-1)$ cycles. A sliding window converts sequential observations into fixed-size vectors, such that they can be classified using a variety of ML algorithms. Figure 3.1 shows an overall flow of detecting JTAG attacks, with data collected using a sliding window. Note that due to the natural variance existing in JTAG operation, the user is not labeled until sufficient evidence has been gathered. The collection of evidence may span several operation cycles.

A central issue concerning the sliding window involves deciding the size of the window, w. A proper window size should be able to capture serial dependence, and make legitimate operation and attacks as separable as possible. A small w performs poorly in separating legitimate operation

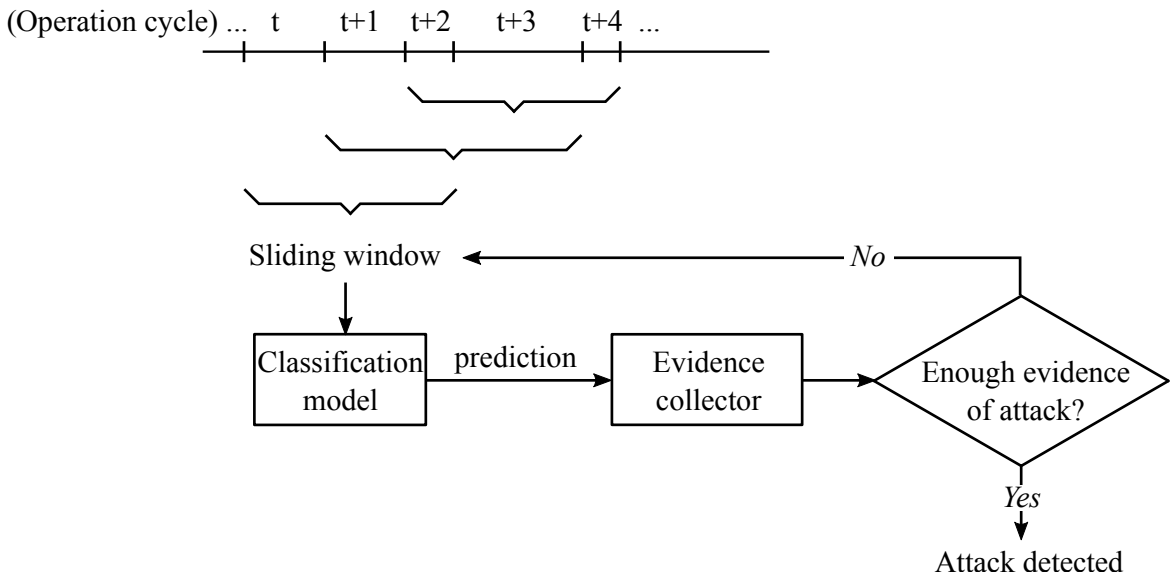


Figure 3.1: The overall flow for online detection of JTAG attacks using a sliding window ($w = 3$).

and attacks, while a large w may lead to poor generalization (i.e., performing well for training data but not for testing data).

The first objective (i.e., capturing serial dependence) can be realized through analyzing representative sequences existing in both legitimate operation and attacks. For example, if the most frequently-observed sequences consist of four operation cycles, then four might be a good choice for the value of w. Alternatively, as indicated by the second objective (i.e., making legitimate operation and attack as separable as possible), we can determine w by checking its influence on classification. Particularly, for a given value of w, the traces of both legitimate operation and attacks are re-collected using a sliding window. Now let us define a window of observation as a multi-dimensional random variable x and the class label as y. Then we can construct two conditional probability distributions for x, namely $P(x|y = L)$ for legitimate operation and $P(x|y = A)$ for attacks. We notice that the more different these two distributions are, the more easily legitimate operation and attacks can be separated. This can be illustrated using Bayes' Theorem. That is, for a given observation x, the probability that x belongs to class y can be determined by the probability

of x given y and the prior probability of y, as shown in Equation 3.1.

$$P(y|x) \propto P(x|y)P(y) \tag{3.1}$$

Since the prior probability is typically fixed, $P(y|x)$ strongly depends on $P(x|y)$. If $P(x|y=A)$ is much greater than $P(x|y=L)$, then we can make a prediction of attack with high confidence. If $P(x|y=A)$ is close to $P(x|y=L)$, then it is hard to make a prediction.

To measure the difference between $P(x|y=L)$ and $P(x|y=A)$, the Kullback-Leibler (KL) divergence, as shown in Equation 3.2, is used. Moreover, to maintain generalization for a large w, the probability distributions are adjusted to $\widetilde{P}(x|y=L)$ and $\widetilde{P}(x|y=A)$ using a regularization term δ, as shown in Equation 3.3 and 3.4. This regularization term can be understood as a uniform prior applied to $P(x|y=L)$ and $P(x|y=A)$.

$$D_{KL}[P(x|y=L)\|P(x|y=A)] = \sum_{X} \widetilde{P}(x=X|y=L) \log \frac{\widetilde{P}(x=X|y=L)}{\widetilde{P}(x=X|y=A)} \tag{3.2}$$

where

$$\widetilde{P}(x=X|y=L) = \frac{P(x=X|y=L)+\delta}{\sum_Z (P(x=Z|y=L)+\delta)} \tag{3.3}$$

$$\widetilde{P}(x=X|y=A) = \frac{P(x=X|y=A)+\delta}{\sum_Z (P(x=Z|y=A)+\delta)} \tag{3.4}$$

The value of δ is set as the reciprocal of the number of observations, because this guarantees that each discrete value of x contains at least one observation. For a small w, δ has negligible impact on the KL-divergence because each value of x is likely to have many observations, while for a large w where most values of x have few observations, the impact of δ becomes significant.

$$\delta = \frac{1}{N_{obs}} \tag{3.5}$$

3.2 Classification Models

In this section, a variety of algorithms, including binary models, one-class models, and recurrent sliding window, are described.

3.2.1 Binary Classifiers

Binary classification aims to construct a model G that maps an observation x to its most probable class label y, based on collected data. The prediction produced by G can be deterministic or probabilistic. A variety of algorithms, including decision tree [65], random forest [66], support vector machines (SVM) [67], artificial neural network (ANN) [68], and k-nearest-neighbor (k-NN) [65], can be used for classification.

$$x \xrightarrow{G} y \qquad x \in \mathbb{R}^d, y \in \{0, 1\} \tag{3.6}$$

A decision tree employs a tree-like structure for classification. As shown in Figure 3.2(a), each node of the decision tree represents a "test" on a feature. A classification starts from the root node, and then descends to its branches according to the "test" outcome; this process terminates at a leaf node where a class label is assigned [65]. The training of a decision tree employs a top-down, greedy search. To avoid overfitting, the set of observations is partitioned into two parts: one for training the full tree, and the other for validating the utility of post-pruning nodes. Subtrees are removed if the resulting pruned tree performs no worse than the original one over the validation set. A random forest involves an ensemble of trees, whose overall prediction is typically based on a majority vote from the individual trees. Each tree in the ensemble is trained using a subset of the data (called bootstrap aggregating or bagging) [66]. The use of bagging represses overfitting and improves model accuracy.

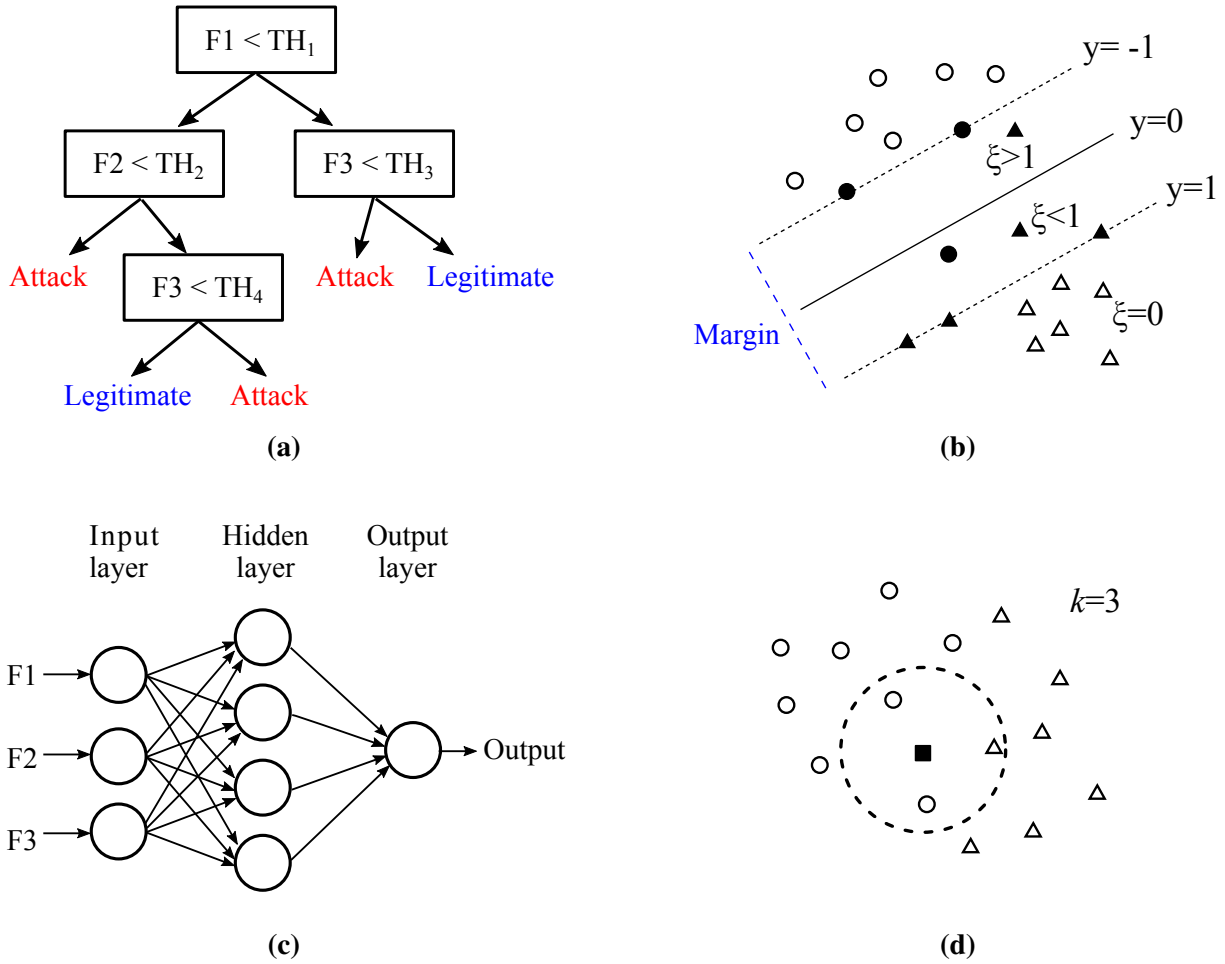


Figure 3.2: Illustration of some example classifiers, including (a) a decision tree, (b) a linear SVM, (c) a three-layer feedforward ANN, and (d) k-NN.

SVMs aim to separate two classes of samples by finding a clear boundary between them, as illustrated in Figure 3.2(b). The training of an SVM involves maximizing the minimum distance between the decision boundary and any of the data samples. The decision boundary is modeled as

$$y(x) = W^T \phi(x) + b \tag{3.7}$$

where x is a sample, $y(x)$ is the prediction of x, and $\phi(x_i)$ represents some feature-space transformation. The decision boundary, characterized by W and b, is determined by solving

$$\underset{W,\xi}{\operatorname{argmin}} \frac{1}{2}\|W\|^2 + C\sum_{i=1}^{N_{obs}} \xi_i \tag{3.8}$$

$$s.t. \qquad \|t_i\left(W^T\phi\left(x_i\right)\right)+b\| \geq 1-\xi_i, \quad i=1,...,N_{obs}$$

where x_i is the i-th training sample, t_i is the true label (either 1 or -1) of x_i, ξ_i is a slack variable that allows samples to be misclassified with a penalty, and $C>0$ controls the trade-off between the slack variable penalty and the margin. This optimization problem is convex so a solution can be derived from its dual representation. More details concerning solving a convex problem can be found in [104]. The decision boundary is formulated as

$$y(x) = \operatorname{sgn}\left(\sum_{i\in S_{SV}} a_i t_i k(x,x_i) + b\right) \tag{3.9}$$

and

$$b = \frac{1}{N_{SV}}\sum_{i\in S_{SV}}\left(t_i - \sum_{j\in S_{SV}} a_j t_j k\left(x_i,x_j\right)\right) \tag{3.10}$$

where S_{SV} denotes the set of support vectors (SVs), N_{SV} represents the number of SVs, and $k(x,x') = \phi(x)\phi(x')$ denotes the kernel function in terms of x and x'.

Multi-layer perceptron (MLP) is a type of feedforward ANN. Figure 3.2(c) shows a MLP with three layers of nodes. Except for the input nodes, each node, named a neuron, involves a nonlinear activation function. A classification starts with supplying a sample to the input nodes; each element of the sample is processed within a neuron, with the result transferred through edges to the next layer. The output neuron finally gives a classification result. An ANN is trained using a technique called backpropagation [65].

k-NN is an instance-based, nonparametric algorithm. It assumes that two observations of the same class are likely close to each other in the hyperspace that the observations reside in. An

observation is labeled by its k nearest neighbors ($k = 3$ in Figure 3.2(d)), through majority voting [105]. A k-NN classifier does not involve a training process; instead, it only requires presence of training data.

3.2.2 Recurrent Sliding Window

The models described in Section 3.2.1 are used for classifying a sliding window. One possible way to improve the classification of a sliding window is to make the classification recurrent, that is to use prior classification predictions as additional input for predicting y_t, i.e., the class label predicted for the window at time t. This might be useful because the prior predictions likely contain information that is however not captured in the current window. For example, in part-of-speech tagging, the grammar of natural language constrains the possible sequences of parts of speech [106]. A simple sliding window cannot capture these patterns unless they are completely manifested in the current input x as well, which is rarely the case. When used for detecting JTAG attacks, a recurrent sliding window may also be useful. Because the previous JTAG instructions are very likely operated by the same user, the earlier predictions provide more evidence of whether the user is legitimate or not. In this work, we only consider the most recent prediction, as shown in Equation 3.11.

$$y_t = G(x, y_{t-1}) \tag{3.11}$$

Note that the recurrent sliding window concept can be used in conjunction with any classifier described in Section 3.2.1.

3.2.3 One-class Models

Different from binary classifiers, one-class models (also named anomaly detectors) are trained using only one class of observations. In this work, the training of a one-class model is based only on legitimate traces. One-class models avoid the use of attacks, which is a strong advantage,

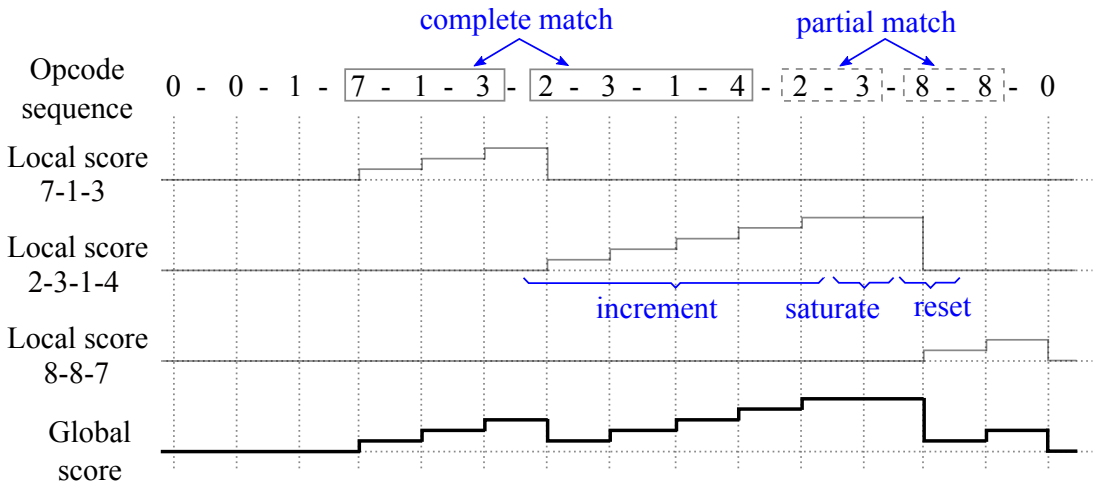


Figure 3.3: Sequence collected in real-time is compared with three representative sequences, namely 7-1-3, 2-3-1-4, and 8-8-7, from the first opcode to the last. Each representative sequence is associated with a local score that may increment, saturate (saturation score is 5 in this example) or reset, according to the matching result. A global score, equal to the maximum local score, serves as an indicator of the security status of the JTAG.

especially considering that collecting all types of attacks is challenging or even impossible. A one-class model (e.g., a one-class SVM) aims to capture regions in the input space where observations form high-density clusters. Then, it determines how probable a given observation is anomalous, based on whether the observation falls within a high-density clusters or not.

In addition to these state-of-the-art algorithms, we develop an anomaly detector based on representative sequences. Legitimate JTAG operation is characterized using a set of representative instruction sequences, such that any deviating operation will be labeled as an attack. The derivation of representative sequences begins with collecting all instruction sequences with different lengths from already-collected legitimate JTAG operations (for example, analysis of legitimate uses of the JTAG reveals that most operations for the OpenSPARC T2 contain three to six instructions). Next, the sequences are refined through 1) removal of low-frequency sequences, 2) checking whether one sequence is contained within another with similar frequency, and if so, removing the shorter subsumed sequence, and 3) removing the sequence with the least impact on detection accuracy iteratively, until the detection accuracy shows an obvious drop.

The security status of the JTAG is measured using a score that indicates whether monitored JTAG operation corresponds to any representative sequence. Specifically, each representative sequence is associated with a score that measures its matching degree with the JTAG operation, with the maximum score selected as a global one. Figure 3.3 illustrates how the global score is calculated. The score associated with each representative sequence (named a local score) is initialized to zero when the chip is powered on. Then opcode sequences are collected in real-time and compared with each representative sequence from the first opcode to the last. Every time an opcode match is observed, the local score increments (or remains the same if it has saturated); otherwise, it is reset to zero. Finally, JTAG operation is labeled as an attack only if the global score stays saturated for more than q consecutive operation cycles. The saturation value and the value of q are predefined and determined empirically from simulation.

Compared to one-class SVMs, the representative-based anomaly detector, incurring moderate overhead, can be used for online detection. Moreover, the representative sequences are derived automatically, which is easier than characterizing JTAG operation using manually-created state machines.

3.3 Delayed Labeling

The prediction made by individual sliding windows might not be reliable, because at times an attack can be similar to or even the same as legitimate operation. This can also be observed in the overlap of legitimate operation and attacks as shown in Figure 2.6 and 2.7. To address this problem, we propose to delay the labeling of JTAG operation as legitimate or illegitimate until sufficient evidence has been collected. Particularly, a hidden Markov model (HMM) is employed for collecting the evidence.

An HMM aims to model a sequential system that is assumed to be a Markov process, i.e., the state at time t only depends on the state at time $t-1$ [104]. Although an HMM assumes that the

state of a system transitions over time (described by the *state transition probabilities*), the state is invisible. To infer the hidden state, one needs to find clues from observations themselves because each state shows a different probability distribution of observations (named *emission probabilities*).

When used for collecting evidence of attacks, the HMM involves two hidden states that represent the actual identification of the user, namely a legitimate state (S_L) and an attack state (S_A). The probabilistic prediction produced by the classifier serves as the observation o of the HMM, i.e., $o = P(y = A|x)$. The hidden state, either S_L or S_A, is inferred based on the sequence of probabilistic predictions. Here, we use $\widetilde{\alpha}_L(t)$ and $\widetilde{\alpha}_A(t)$ to represent the likelihood that the hidden state at time t is S_L and S_A, respectively, given the observation sequence $o_1 o_2 \ldots o_t$ and the parameters of the HMM, λ. $\widetilde{\alpha}_L(t)$ and $\widetilde{\alpha}_A(t)$ are represented in Equations 3.12 and 3.13.

$$\widetilde{\alpha}_L(t) = P(s_t = S_L \mid o_1 o_2 \ldots o_t, \lambda) \tag{3.12}$$

$$\widetilde{\alpha}_A(t) = P(s_t = S_A \mid o_1 o_2 \ldots o_t, \lambda) \tag{3.13}$$

For example, if the prediction produced by the classifier is a very positive value (i.e., more likely an attack), then $\widetilde{\alpha}_A(t)$ will be increased compared to $\widetilde{\alpha}_A(t-1)$, which can be understood as an accumulation of evidence for a JTAG attack. This may not label the operation as an attack immediately, but the labeling will happen if $\widetilde{\alpha}_A(t)$ exceeds a predefined threshold δ_{ATK} (i.e., sufficient evidence has been collected). The value of δ_{ATK} indicates how strict the criterion is to label the user as an attacker. A small δ_{ATK} represents a harsh criterion that may classify more legitimate operation as attacks (i.e., false positive), while a large δ_{ATK} indicates a tolerant criterion that may classify more attacks as legitimate (i.e., false negative). The impact caused by false positive and false negative typically depends on the underlying problem. Therefore, the optimal value for δ_{ATK} should also be determined based on specific objectives for preventing false positives and false negatives, which is a subject for future work.

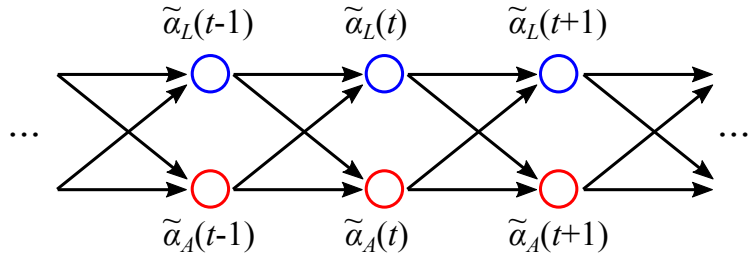


Figure 3.4: The computation of $\widetilde{\alpha}_L(t)$ and $\widetilde{\alpha}_A(t)$ involves a lattice-like process.

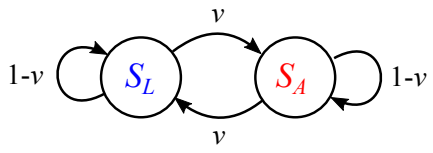
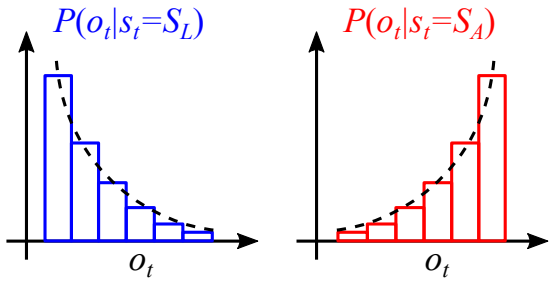
Initial probability	$P(s_1{=}S_L) = \pi$ $P(s_1{=}S_A) = 1\text{-}\pi$
State transition probability	
Emission probability	

Figure 3.5: The parameters of an HMM used for collecting evidence of an attack. The hidden state always starts from S_L, i.e., assuming the user as legitimate initially. State transition, either from S_L to S_A or from S_A to S_L, occurs with a small probability v. The emission probability can be derived through analyzing the probabilistic predictions produced for legitimate operation and attacks, respectively. More precisely, legitimate operation is more likely to produce a negative prediction that results in a small o_t, while an attack is more likely to produce a positive prediction that results in a large o_t.

Assuming N hidden states and M observation values, the parameters of the HMM (λ) include initial probabilities (π), state transition probabilities (R), and emission probabilities (B), as shown in Equations 3.14-3.17.

$$\lambda = (R, B, \pi) \tag{3.14}$$

where

$$R = \{r_{ij}\}, \qquad r_{ij} = P(s_{t+1} = S_j \mid s_t = S_i) \quad 1 \le i, j \le N \tag{3.15}$$

$$B = \{b_{im}\}, \qquad b_{im} = P(o_t = v_m \mid s_t = S_i) \quad 1 \le i \le N, 1 \le m \le M \tag{3.16}$$

$$\pi = \{\pi_i\}, \qquad \pi_i = P(s_1 = S_i) \quad 1 \le i \le N \tag{3.17}$$

The value of M depends on how many discrete values are used for approximating the probabilistic prediction. In this work, $N = 2$ since there are two hidden states, and $M = 8$. Then, $\alpha_L(t)$ and $\alpha_A(t)$ can be represented as

$$\alpha_L(t) = P(o_1 o_2 \ldots o_t, s_t = S_L \mid \lambda) \tag{3.18}$$

$$\alpha_A(t) = P(o_1 o_2 \ldots o_t, s_t = S_A \mid \lambda) \tag{3.19}$$

$\alpha_L(t)$ and $\alpha_A(t)$ can be derived recursively using a forward procedure [104]. For $t = 1$,

$$\alpha_L(1) = \pi_L b_{Lo_1} \quad \alpha_A(1) = \pi_A b_{Ao_1} \tag{3.20}$$

and for $t > 1$,

$$\alpha_L(t) = [\alpha_L(t-1) r_{LL} + \alpha_A(t-1) r_{AL}] \cdot b_{Lo_t} \tag{3.21}$$

$$\alpha_A(t) = [\alpha_L(t-1) r_{LA} + \alpha_A(t-1) r_{AA}] \cdot b_{Ao_t} \tag{3.22}$$

Then, $\widetilde{\alpha}_L(t)$ and $\widetilde{\alpha}_A(t)$ can be derived through normalizing $\alpha_L(t)$ and $\alpha_A(t)$.

$$\widetilde{\alpha}_L(t) = \frac{\alpha_L(t)}{\alpha_L(t) + \alpha_A(t)} \tag{3.23}$$

$$\widetilde{\alpha}_A(t) = \frac{\alpha_A(t)}{\alpha_L(t) + \alpha_A(t)} \tag{3.24}$$

The computation of $\alpha_L(t)$ and $\alpha_A(t)$ involves a lattice-like process, as shown in Figure 3.4.

Figure 3.5 shows the parameters of the HMM. The initial probability depends on the prior assumption of the user. In this work, we initially assume the user is legitimate, i.e., $\pi = 1$, although π can also be set to a smaller value between 0 and 1. The state transition probability, either from state S_L to state S_A or from S_A to S_L, occurs with a small value, v, and thus describes how likely the identity of the user transitions between legitimacy and illegitimacy. The transition might occur in many scenarios. For example, a board discarded due to malfunction/upgrade might be exploited by an attacker for malicious purposes. An attacker may even purchase a brand-new board for malicious purposes. Although it is hard to find a precise value for v, we heuristically set it to a small number (i.e., 0.1) because the same user is likely to continue operating the JTAG. The emission probability can be derived through analyzing the probabilistic predictions produced for legitimate operation and attacks, respectively. More precisely, legitimate operation is more likely to produce a negative prediction that results in a small o_t, while an attack is more likely to produce a positive prediction that results in a large o_t.

3.4 Sequence Models

As explained in Section 3.1, sequence models can learn serial dependence existing in time-based data better than sliding-window-based detectors. Even though sequence models are typically complex and not suitable for online detection, they are still evaluated in this work, including recurrent neural networks, HMMs, and hierarchical HMMs. In this work, we only consider the instruction opcode as the feature processed by the sequence models.

3.4.1 Recurrent Neural Network

Recurrent neural networks (RNNs) form a class of artificial neural networks that employ neuron feedback. In feedforward-only networks, input samples are fed to network inputs, and transferred in one direction towards the output layer. For RNNs, the state of a neuron (c_t) depends not only on

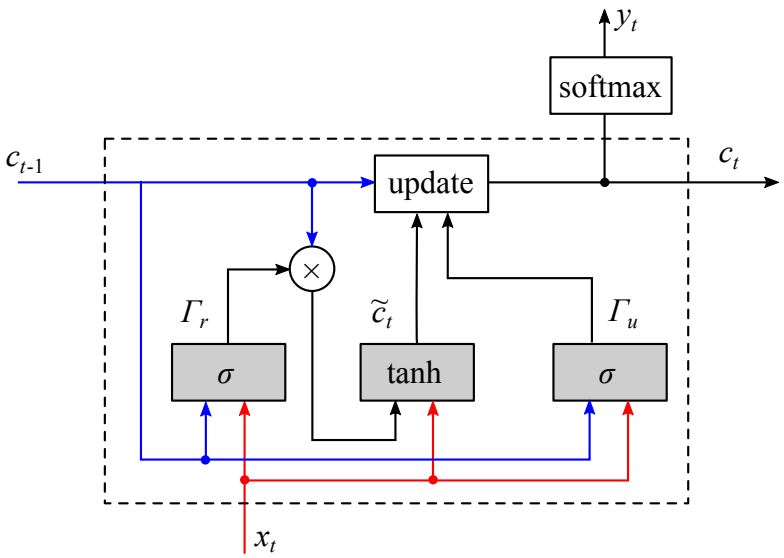


Figure 3.6: A gated recurrent unit (GRU).

the current input sample (x_t), but also on what they have perceived previously (c_{t-1}). RNNs differ from feedforward networks due to the feedback loop involving their past decisions. This feedback provides RNNs with memory, enabling sequential information to be learned.

$$c_t = f(c_{t-1}, x_t) \tag{3.25}$$

The function $f(\cdot)$ is modeled using a gated recurrent unit (GRU) as shown in Figure 3.6 and described by equations 3.26-3.29. Compared to the LSTM, the GRU is more computationally efficient. The GRU employs a memory cell, denoted by c_t. At time t, c_t is updated based on the previous memory cell c_{t-1} and a candidate $\widetilde{c}_t$. Γ_u serves as a coefficient that decides the impact of c_{t-1} and $\widetilde{c}_t$ on c_t. The calculation of $\widetilde{c}_t$ involves a coefficient Γ_r that represents the relevance of c_{t-1} relative to the current observation x_t. The training of an RNN also employs a backpropagation technique (referred to as *backpropagation through time*), which however requires the feedback of the GRU to be unfolded. Finally, the output y_t involves a softmax function to c_t, since the attack

detection is a classification task.

$$\widetilde{c_t} = \tanh(W_c \cdot [\Gamma_r c_{t-1}, x_t] + b_c) \tag{3.26}$$

$$\Gamma_u = \sigma(W_u \cdot [c_{t-1}, x_t] + b_u) \tag{3.27}$$

$$\Gamma_r = \sigma(W_r \cdot [c_{t-1}, x_t] + b_r) \tag{3.28}$$

$$c_t = \Gamma_u \cdot \widetilde{c_t} + (1 - \Gamma_u) \cdot c_{t-1} \tag{3.29}$$

3.4.2 Hidden Markov Model

HMMs are another model widely used for sequential analysis. The basics of HMM have been described in Section 3.3. When used for analyzing JTAG instruction sequences, an HMM is trained using observed JTAG traces. As described by equation 3.30. the training of an HMM aims to find the most likely parameters that maximize the probability of the observations. The training employs an expectation-maximization (EM) approach [104]. In this work, to detect attacks, two HMMs are trained, based on legitimate and attack traces, respectively. Then, an observed sequence is evaluated using both HMMs, and labeled based on the outcome with higher probability.

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmax}} P(o_1 o_2 \ldots o_t \mid \lambda) \tag{3.30}$$

HMMs assume conditional independence for observations given the state. That is, the current observation only depends on the current state, but does not depend explicitly on previous observations. For example, if a state represents an MBIST operation, then the use of each MBIST instruction is independent given the state. However, this assumption might be problematic because these instructions are typically executed in specific orders. To mitigate this problem, standard HMMs are extended to a hierarchical structure.

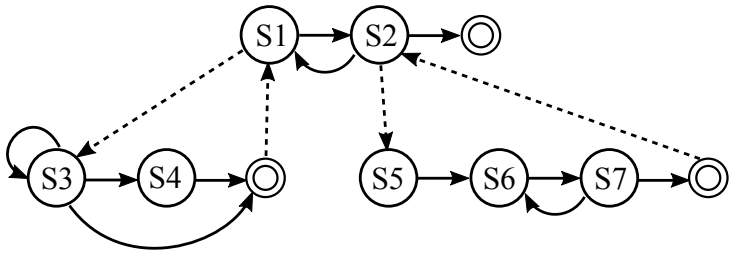


Figure 3.7: The architecture of a two-level HHMM.

Hierarchical HMMs (HHMMs) are structured multi-level stochastic processes [107]. HHMMs generalize the standard HMMs by making each of the hidden states an "autonomous" probabilistic model on its own, such that the states of an HHMM emit sequences rather than a single symbol. Here, we consider an HHMM with two levels of states, as illustrated in Figure 3.7. The main states (i.e., in the first level) typically represent JTAG functions, such as MBIST and cache access, while the sub-states (i.e., in the second level) describe how instructions execute for each JTAG function. For the HHMM in Figure 3.7, state S1 may activate its sub-state S3 (through a vertical transition), which then lead to state transitions among S3 and S4 according to the corresponding transition probabilities (via horizontal transitions). Upon completion of horizontal transitions, control returns to the main state that originates the activation (i.e., S1). Next, the control may transition to S2, or activate its sub-states (i.e., S3 and S4) again. Note that only the sub-states (i.e., S3-S4, S5-S7) emit observations, while the main states (i.e., S1-S2) do not emit observations directly.

The training of the two-level HHMM involves training of the main HMM and the sub-HMMs. To train sub-HMMs, we first extract representative sequences from legitimate JTAG operation (for example, using the method described in Section 3.2.3). These representative sequences are then divided into K clusters using K-means, each cluster used for training a sub-HMM (shown in Algorithm 1). Now, it is obvious that the main HMM has K states, each one corresponding to a sub-HMM. To derive the state transition probabilities for the main HMM, we need to tag the representative sequences within legitimate operation, and then count the transitions between

Algorithm 1: Use K-means to cluster representative sequences and train an HMM for each cluster

Input : A set of representative sequences $\{s_i\}$
K HMMs whose parameters $\lambda_1, \lambda_2, \ldots, \lambda_K$ are randomly initialized
Output: K trained HMMs, i.e., $\lambda_1, \lambda_2, \ldots, \lambda_K$

```
do
    for each representative sequence s_i do
        l = argmax_{k=1,2,...,K} hmm_decode(s_i, λ_k)
        assign s_i to the l-th cluster C_l
    end
    for k ← 1 to K do
        train the k-th HMM (λ_k) based on the k-th cluster of representative sequences (C_k)
    end
while λ_1, λ_2, ..., λ_K converge;
```

them. Note that in Algorithm 1, the function hmm_decode(s, λ) calculates the posterior state probabilities for the sequence s based on a hidden Markov mode λ.

Attack detection using an HHMM is similar to an HMM. That is, two HHMMs are trained for legitimate operation and attacks, respectively. An observed sequence is evaluated using both HHMMs and labeled based on the outcome with higher probability.

3.5 Detection of Unknown Attacks

ML models can naturally identify unseen data, which however is based on the assumption that the data are from the same distribution as the training data. If the data are from a different distribution, then the effectiveness of ML models is weakened. For example, if the attacker exploits new attack strategies or targets an IC component that was not considered when building the ML model, then the ML model may fail to detect such attacks. Figure 3.8 illustrates a new type of attack as a cluster that is far from both legitimate operation and known attacks. Even though the binary classifier will finally label each new attack instance as either positive or negative, its accuracy is poor. This inaccuracy can be modeled using the notion of *open-space risk* introduced in [71]. An open space refers to an area where few prior observations reside. Considering that the

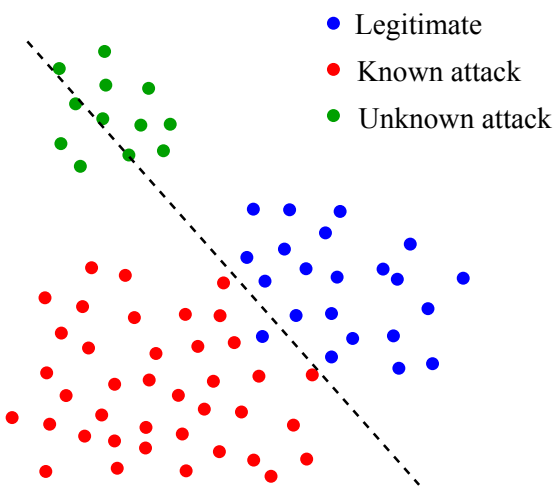


Figure 3.8: A binary classifier suffers from open-space risk, i.e., unable to correctly classify observations that are "far" from historically-observed data (e.g., legitimate operation and known attacks).

collection of legitimate operation is relatively complete compared to attacks (i.e., a legitimate user typically knows well how to operate the JTAG), it is safer to label observations located in an open space as a new type of attack.

To minimize open-space risk, we develop a cascade model that combines a one-class model and a binary classifier, as shown in Figure 3.9. One-class models are preferred for anomaly detection, while binary classifiers are more effective in determining the boundary between different classes. Specifically, the method of Weibull-calibrated SVM described in [108] is employed. The work in [108] demonstrates that the Weibull-calibrated SVM performs significantly better than common binary models, such as SVM, Logistic Regression, and Nearest Neighbor. The first step involves training a one-class SVM using only legitimate traces. Then, the probability of class inclusion for the one-class SVM is modeled by fitting a Weibull distribution based on all training samples. This process converts the distance between an observation and the classification boundary to a probability $P_O(x)$. The next step is to train a binary SVM using both legitimate and attack traces. Then, two Weibull distributions are fitted based on legitimate and attack samples, respectively (both correctly-classified and misclassified samples are used for fitting the distributions). Particularly, these two distributions model the probability that an observation x is legitimate, $P_L(x)$, and the

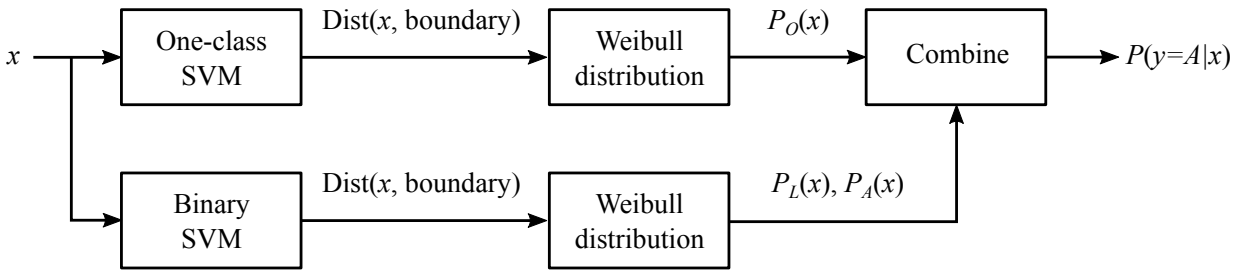


Figure 3.9: A cascade model is constructed by combining a one-class SVM and a binary SVM for detecting unknown attacks.

probability that the observation is attack, $P_A(x)$. Note that $P_L(x) + P_A(x)$ is not necessarily equal to one because $P_L(x)$ and $P_A(x)$ are derived from two Weibull distributions. For example, if x resides in an open space, then both $P_L(x)$ and $P_A(x)$ are likely small.

The final prediction $P(y = A|x)$ is based on $P_O(x)$, $P_L(x)$ and $P_A(x)$. First, a binary variable ι_L is defined, such that $\iota_L = 1$ if $P_O(x) > \lambda_\tau$, and $\iota_L = 0$ otherwise. λ_τ is a threshold that determines the degree of anomaly. Typically, λ_τ is set to a very small probability such that only very anomalous samples are identified as attacks. As suggested by [108], λ_τ is set to 0.0001 in the experiments of this work. Then, the probabilistic prediction of an attack is calculated as

$$P(y = A|x) = 1 - \sqrt{P_L(x) \times [1 - P_A(x)] \times \iota_L} \tag{3.31}$$

According to Equation 3.31, if an observation x is too far from legitimate operation, then ι_L is likely zero, meaning that x is identified as an unknown attack with high probability. If x resides in a cluster of legitimate operation, then both $P_L(x)$ and $1 - P_A(x)$ are large, resulting in a small $P(y = A|x)$. If x resides in a cluster of known attack, then both $P_L(x)$ and $1 - P_A(x)$ are small, resulting in a large $P(y = A|x)$. If x resides in an open area, then $P_L(x)$ is likely small and $1 - P_A(x)$ is likely large, resulting in a medium $P(y = A|x)$. Note that $P(y = A|x)$ needs to be supplied to the evidence collector described in Section 3.3.

3.6 Experiments

This section evaluates the performance of attack detection for all of the models described in the earlier sections of this chapter. Same as Chapter 2, the experiments are based on the OpenSPARC T1 and T2 processors that are developed by Oracle [30, 64], and the LEON3 processor that is developed by Gaisler [28].

3.6.1 Search for Window Size

To find the optimal value of w, we evaluate the impact of w on the regularized KL-divergence for the OpenSPARC T1, T2, and the LEON3 processors, with results shown in Figure 3.10. For the OpenSPARC T1 and T2, the features shown in Figure 2.5 are used, while for the LEON3, nine features are used, four for BUS_ADDR, four for BUS_DATA, and one for BUS_RW. According to the results shown in Figure 3.10, as w changes, the divergence between legitimate JTAG operation and attacks increases when w is small, but then saturates and even decreases when w is large, due to the regularization effect provided by δ. For the OpenSPARC T1 and T2, a w of four or five performs best. However, for the LEON3, the optimal value of w is almost 40, revealing that the bus operation of the LEON3 is not only more complex than the operation of JTAG TAP, but is much more difficult to classify.

3.6.2 Sliding-window-based Detectors

In this set of experiments, the capability of detecting known attacks is evaluated for the sliding-window-based detectors and the representative-based anomaly detection. In particular, for binary classifiers, five-fold cross validation is performed, i.e., all traces, including legitimate ones and attacks, are partitioned into five subsets of the same size; four subsets are used for training the classifier, and the fifth for evaluating the accuracy of the resulting classifier; each subset is used once for testing, with the final accuracy being averaged. For one-class models and the representative-based

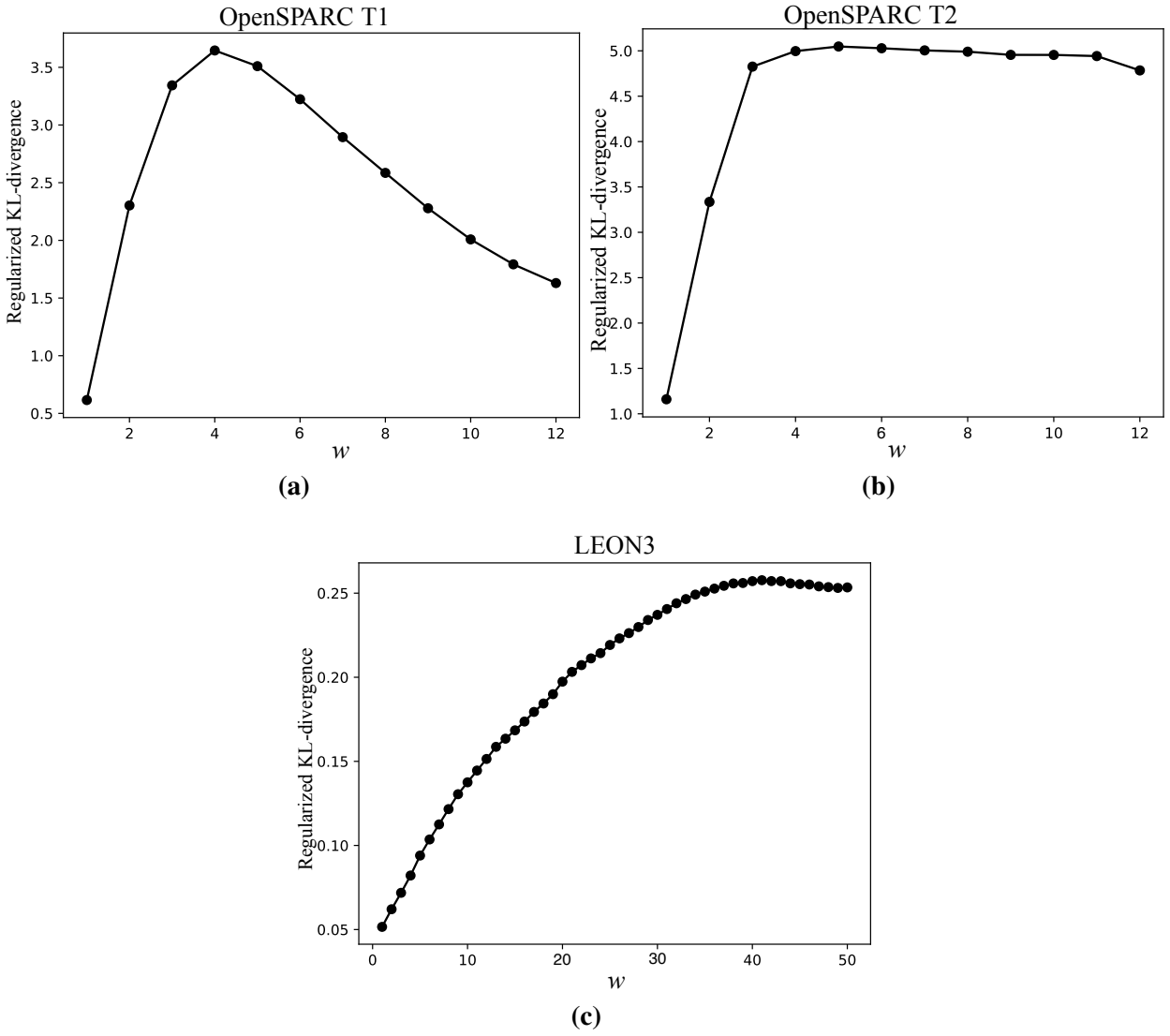


Figure 3.10: The impact of window size (w) on the regularized KL-divergence between legitimate JTAG operation and attacks, for (a) the OpenSPARC T1, (b) the OpenSPARC T2, and (c) the LEON3.

anomaly detector, 90% of the legitimate traces are used for training the model, and the remaining legitimate traces and attack traces are used for testing the accuracy of the model.

Table 3.1 lists the parameters selected for each classifier. The performance is evaluated using three metrics, namely error rate, false positive rate (FPR), and false negative rate (FNR) as defined

TABLE 3.1: THE PARAMETERS SELECTED FOR VARIOUS DETECTION MODELS.

Algorithm	Setting
Decision tree	max_depth = 16
Random forest	n_trees = 3, max_depth = 10
SVM	kernel = RBF, $\gamma = 0.125$
MLP	n_hidden_layer = 1, n_hidden_neuron = 10, learning_rate = 0.02, max_epochs = 100
k-NN	$k = 3$
Naïve Bayes	model = Multinomial
One-class SVM	$\nu = 0.05$, kernel = RBF, $\gamma = 0.125$
Representative	saturation_score = 5, $q = 25$

in Equations 3.32-3.34, respectively.

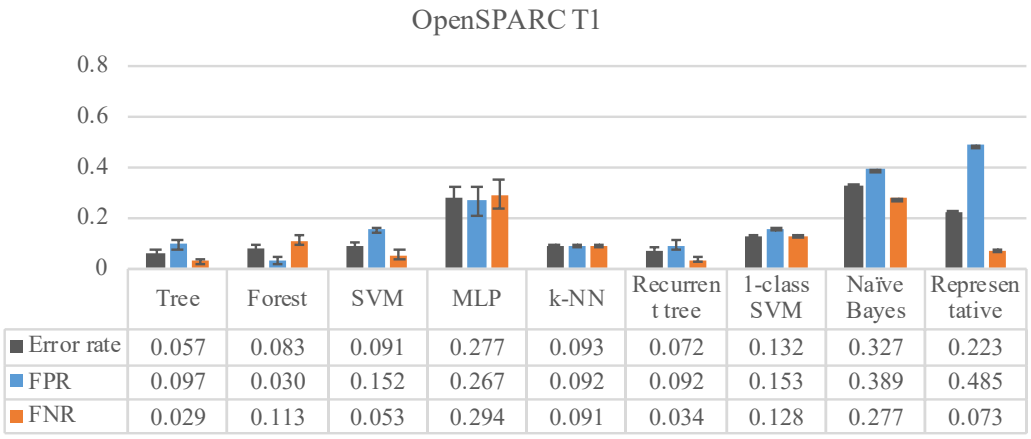
$$\text{Error rate} = \frac{\text{No. of misclassified traces}}{\text{No. of all traces}} \tag{3.32}$$

$$\text{FPR} = \frac{\text{No. of legitimate traces that are classified as attack}}{\text{No. of legitimate traces}} \tag{3.33}$$

$$\text{FNR} = \frac{\text{No. of attack traces that are classified as legitimate}}{\text{No. of attack traces}} \tag{3.34}$$

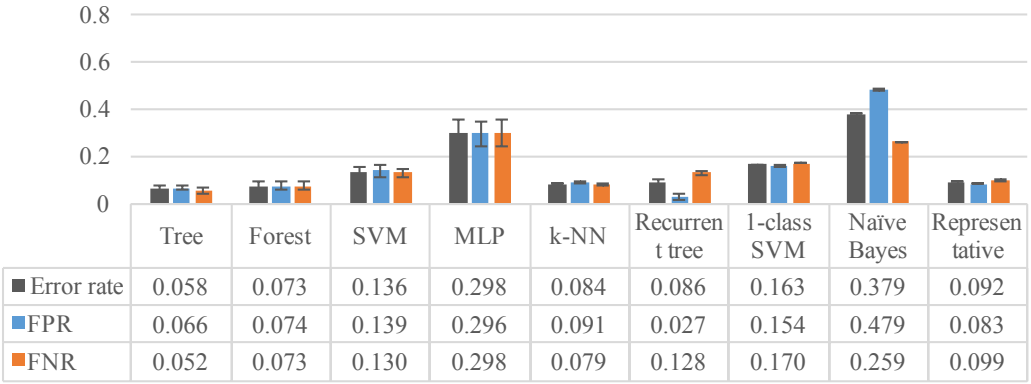
In addition, the threshold for the evidence collector (i.e., δ_{ATK}) is set to 0.9, which leads to balanced FPR and FNR (nevertheless, δ_{ATK} can be set to other values to fit the needs specified by the designer). Alternatively, the performance of classifiers can be evaluated using the receiver operating characteristic (ROC) curve and measured using the area under the curve (AUC) [109]. Different from FPR and FNR, AUC can evaluate classifier performance regardless of threshold. Note that the parameters listed in Table 3.1 are selected because they demonstrate better error rate than others through simulation.

The results of each classifier is exhibited in Figure 3.11, where the error bars represent the standard deviation resulting from cross validation. Figure 3.11 show that the decision tree, random forest, SVM, and k-NN have similar performance in identifying attacks for all three benchmarks. The MLP shows poor performance for the OpenSPARC T1 and T2, possibly due to insufficient



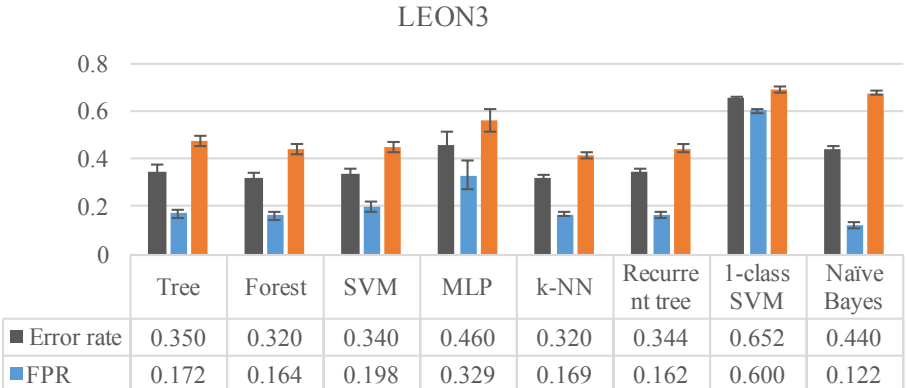
	Tree	Forest	SVM	MLP	k-NN	Recurrent tree	1-class SVM	Naïve Bayes	Representative
Error rate	0.057	0.083	0.091	0.277	0.093	0.072	0.132	0.327	0.223
FPR	0.097	0.030	0.152	0.267	0.092	0.092	0.153	0.389	0.485
FNR	0.029	0.113	0.053	0.294	0.091	0.034	0.128	0.277	0.073

(a)



	Tree	Forest	SVM	MLP	k-NN	Recurrent tree	1-class SVM	Naïve Bayes	Representative
Error rate	0.058	0.073	0.136	0.298	0.084	0.086	0.163	0.379	0.092
FPR	0.066	0.074	0.139	0.296	0.091	0.027	0.154	0.479	0.083
FNR	0.052	0.073	0.130	0.298	0.079	0.128	0.170	0.259	0.099

(b)



	Tree	Forest	SVM	MLP	k-NN	Recurrent tree	1-class SVM	Naïve Bayes
Error rate	0.350	0.320	0.340	0.460	0.320	0.344	0.652	0.440
FPR	0.172	0.164	0.198	0.329	0.169	0.162	0.600	0.122
FNR	0.479	0.441	0.449	0.561	0.414	0.442	0.691	0.675

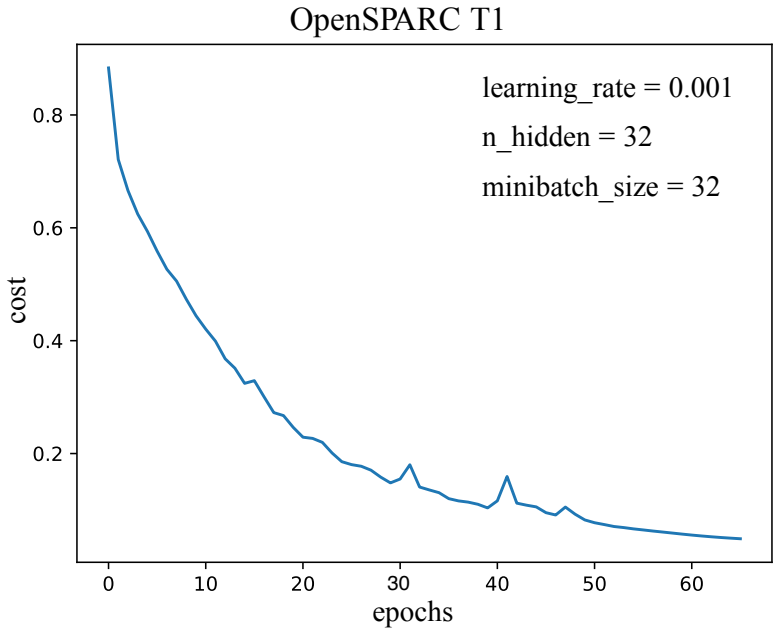
(c)

Figure 3.11: Sliding-window-based detectors and the representative-based anomaly detector are evaluated using five-fold cross validation, with the performance evaluated using error rate, FPR, and FNR.

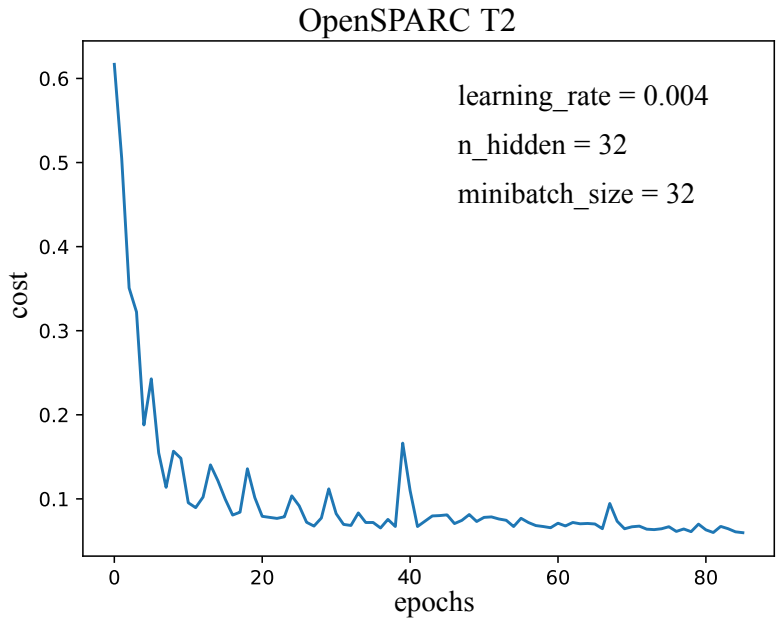
number of hidden neurons. The recurrent tree does not show obvious improvement compared to the decision tree, meaning that the most recent binary prediction does not improve attack detection. The results also demonstrate that one-class SVM and the representative-based anomaly detector are inferior to binary classifiers in identifying decision boundaries. Moreover, the LEON3 processor exhibits much higher error rate than the OpenSPARC T1 and T2, revealing that detecting anomalies in bus traffic is more challenging than monitoring the JTAG TAP. Thus, more features and more complex models are required for more accurate classification. Note that the one-class SVM results in error rate that is greater than 0.5, as exhibited in Figure 3.11(c). In this case, flipping the classification results can improve the accuracy.

3.6.3 Sequence Models

In this set of experiments, the performance of sequence models, including RNNs, HMMs and HHMMs, are evaluated. Note that only the feature OPCODE is considered. In addition, all JTAG traces are partitioned into three sets, namely, a training set (80%), a validation set (10%) and a testing set (10%). The training of the RNN employs a cost function of softmax-with-cross-entropy. Figure 3.12 shows that the cost is decreasing as the training proceeds, for both the OpenSPARC T1 and T2. The hyper-parameters, including learning rate, number of hidden neurons, size of mini-batch, are searched using an exhaustive, coarse-to-fine method. Note that the number of hidden neurons decides the history capacity that the RNN can memorize, while the learning rate decides the speed and smoothness of convergence. To avoid overfitting, accuracy of the validation set is examined after each epoch, such that training stops when the accuracy for the validation set starts to decrease. Finally, the test set is used for evaluating the performance of the trained model. According to the results shown in Figure 3.13, the RNN demonstrates an error rate of 1.3% for the OpenSPARC T1 and 2.2% for the OpenSPARC T2, which is much better than the sliding-window-based detectors. A possible explanation is that an RNN is more effective in learning serial dependence and retaining a varying amount of historic information.



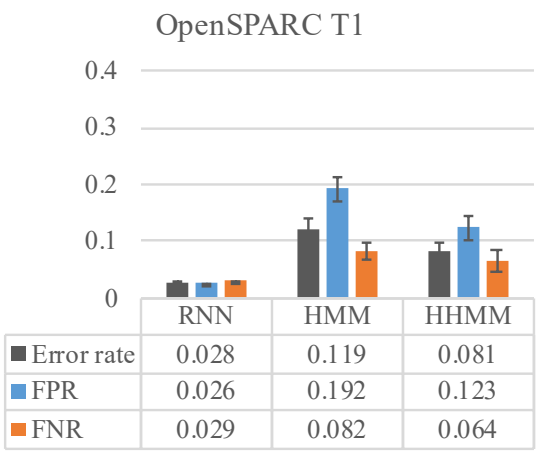
(a)



(b)

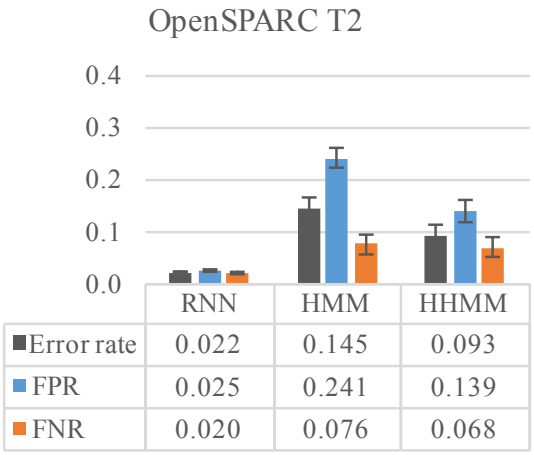
Figure 3.12: The cost decreases as the training of an RNN proceeds.

Next, two HMMs are trained for legitimate and attack traces, named HMM-X and HMM-Y, respectively. Each testing trace is evaluated using both HMMs, with the log-probability plotted in Figure 3.14. Since a trace is labeled by the HMM that gives a higher probability, the dots located



	RNN	HMM	HHMM
Error rate	0.028	0.119	0.081
FPR	0.026	0.192	0.123
FNR	0.029	0.082	0.064

(a)



	RNN	HMM	HHMM
Error rate	0.022	0.145	0.093
FPR	0.025	0.241	0.139
FNR	0.020	0.076	0.068

(b)

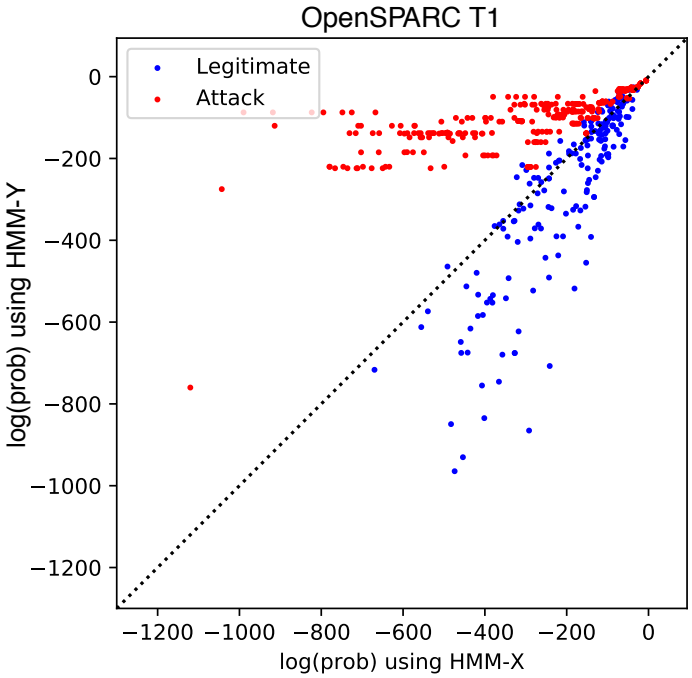
Figure 3.13: Sequence models, including an RNN, an HMM, and a two-level HHMM, are evaluated using error rate, FPR, and FNR.

in different sides of the diagonal line will have different labels. Figure 3.14 shows that many dots are in the wrong side. Then, a two-layer HHMM is trained. The main HMM has twelve states, and each sub-HMM has three states. Figure 3.13 shows that the HHMM improves the performance of the HMM, but is still inferior to the RNN.

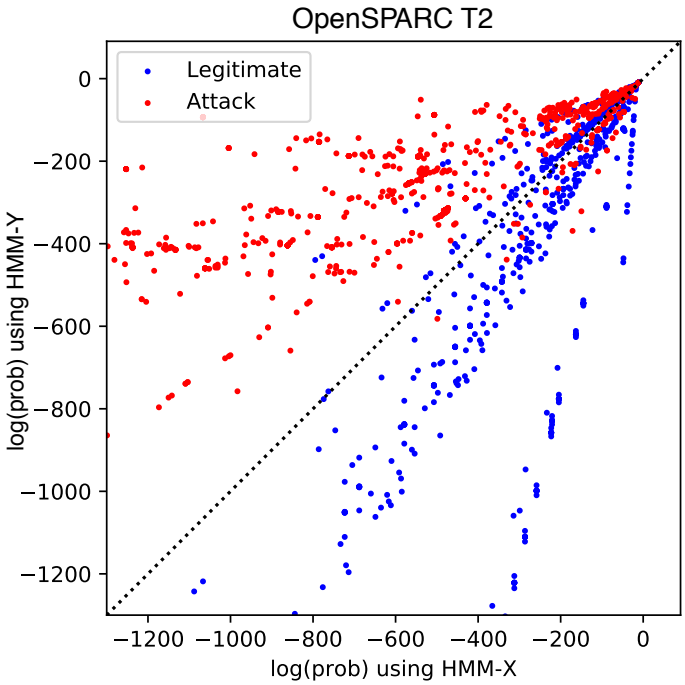
3.6.4 Detection of Unknown Attacks

In this set of experiments, the performance of detecting unknown JTAG attacks is evaluated for the OpenSPARC T2. An unknown JTAG attack is one that targets a different IC component or exploits a different strategy not included in training the model. In the first case, the attack traces are divided into eight categories based on the components they target[1]. Seven out of eight categories of attacks and all cases of legitimate operations are simulated using five-fold cross-validation, with the FPR and the FNR evaluated. Then the eighth category of attack is evaluated using the trained classifier, with the error rate evaluated.

[1]The components include clock control, control register, electronic fuse, L2 cache access, logic BIST, memory BIST, and shadow scan. Note that attack0 represents an initial search of all instructions and data registers, not referring to a specific component but still considered as a category of attack.



(a)



(b)

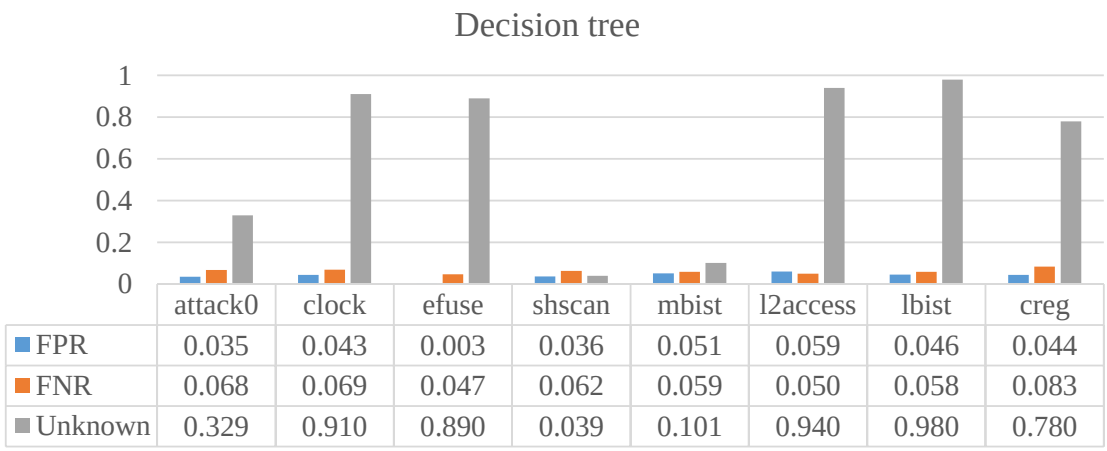
Figure 3.14: Two HMMs, namely HMM-X and HMM-Y, are trained based on legitimate JTAG operation and attacks, respectively. Each testing trace is evaluated using both HMMs, with the log-probability plotted.

In the second case, the attack traces are divided into nine categories based on the strategies listed in Table 2.2, and then evaluated using the similar method for the first case. The performance of the cascade model is compared with a decision tree and an RNN, as shown in Figure 3.15 and 3.16. According to the results, although performing better than a decision tree, the cascade model is not able to detect all categories of unknown attacks effectively. A possible explanation involves that some categories of attacks, overlapping with legitimate operation, are hard to detect. To illustrate this, Figure 3.17 shows two examples of unknown attacks (targeting an IC component not included in training) for the OpenSPARC T2. The first example of attack resides in an open space and thus can be detected by the one-class model, while the second example of attack overlaps with legitimate operation, which is hard to classify using both one-class and binary classifiers. Note that the background contours outline the boundary of the one-class SVM, where the white area represents inlier and the blue area represents outlier. Figure 3.15(c) and 3.16(c) show that an RNN can also improve performance of detection unknown attacks. However, there are still several categories of attacks that cannot be detected by both the cascade model and the RNN. Considering the improvement resulting from the cascade model and the RNN, a possible way of further improving the performance is to construct the cascade model using a one-class classifier and an RNN, a good subject for future work.

3.7 Discussion

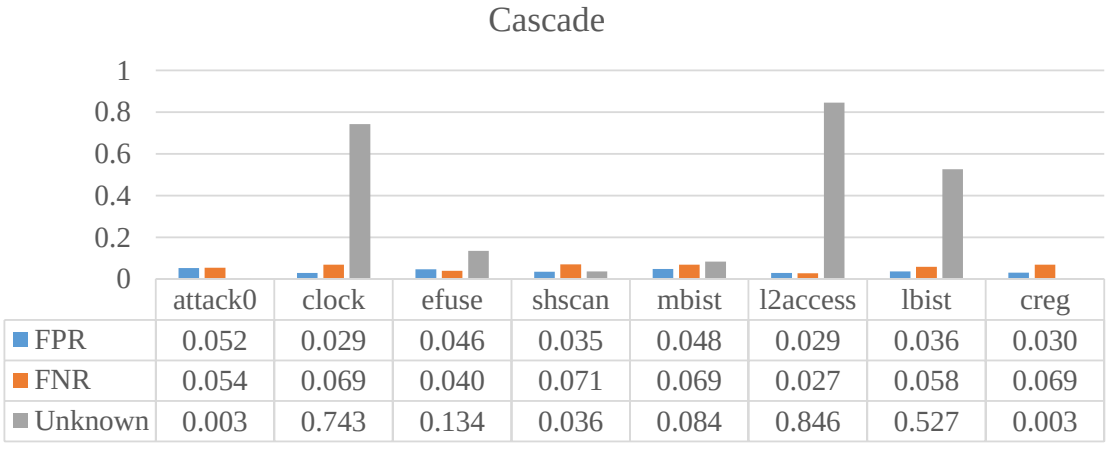
This section discusses two issues, namely ability of detecting unknown attacks and occurrence of false positives.

Collecting and evaluating unknown attacks are not trivial tasks. First, the "unknown" attacks used in Section 3.6.4 are not perfectly unknown. This is because different categories of attacks might be similar to some extent. For example, attacks of different IC components, although achieved by different sets of JTAG instructions, exploit similar strategies as listed in Table 2.2. Second, the experiment results exhibited in Figure 3.15 do not guarantee that the detectors can



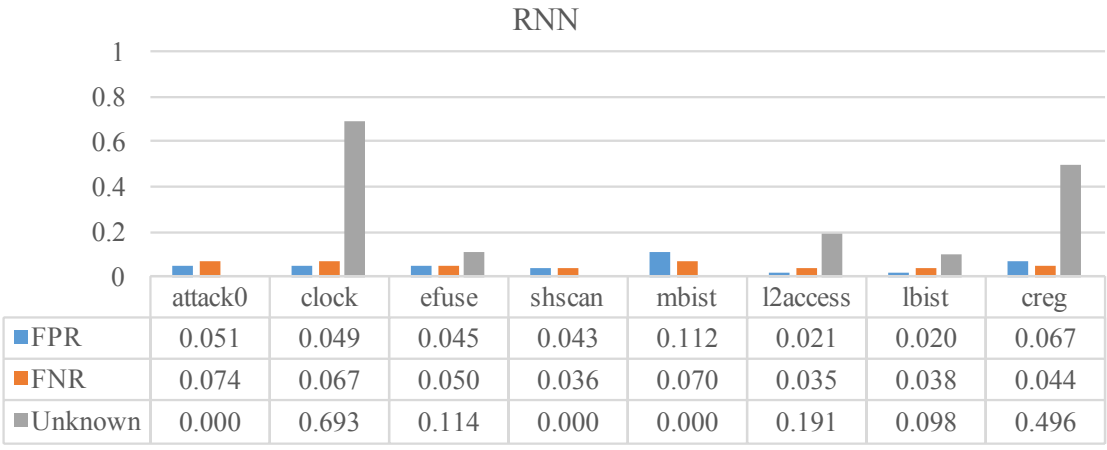
	attack0	clock	efuse	shscan	mbist	l2access	lbist	creg
FPR	0.035	0.043	0.003	0.036	0.051	0.059	0.046	0.044
FNR	0.068	0.069	0.047	0.062	0.059	0.050	0.058	0.083
Unknown	0.329	0.910	0.890	0.039	0.101	0.940	0.980	0.780

(a)



	attack0	clock	efuse	shscan	mbist	l2access	lbist	creg
FPR	0.052	0.029	0.046	0.035	0.048	0.029	0.036	0.030
FNR	0.054	0.069	0.040	0.071	0.069	0.027	0.058	0.069
Unknown	0.003	0.743	0.134	0.036	0.084	0.846	0.527	0.003

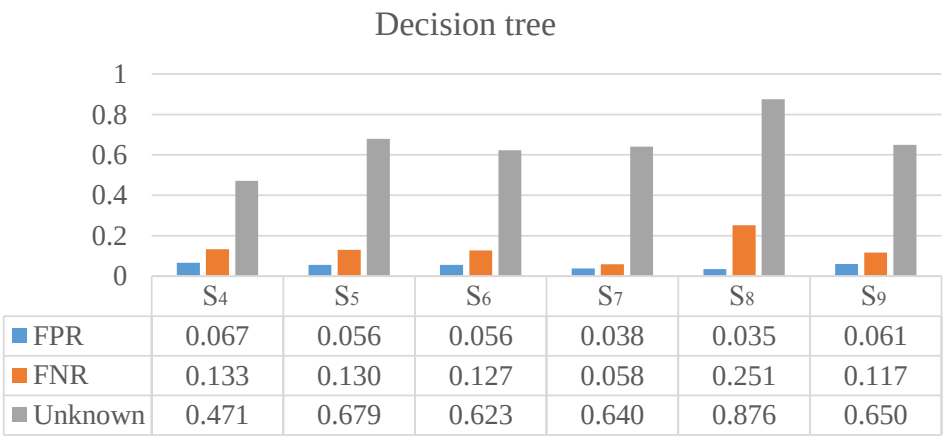
(b)



	attack0	clock	efuse	shscan	mbist	l2access	lbist	creg
FPR	0.051	0.049	0.045	0.043	0.112	0.021	0.020	0.067
FNR	0.074	0.067	0.050	0.036	0.070	0.035	0.038	0.044
Unknown	0.000	0.693	0.114	0.000	0.000	0.191	0.098	0.496

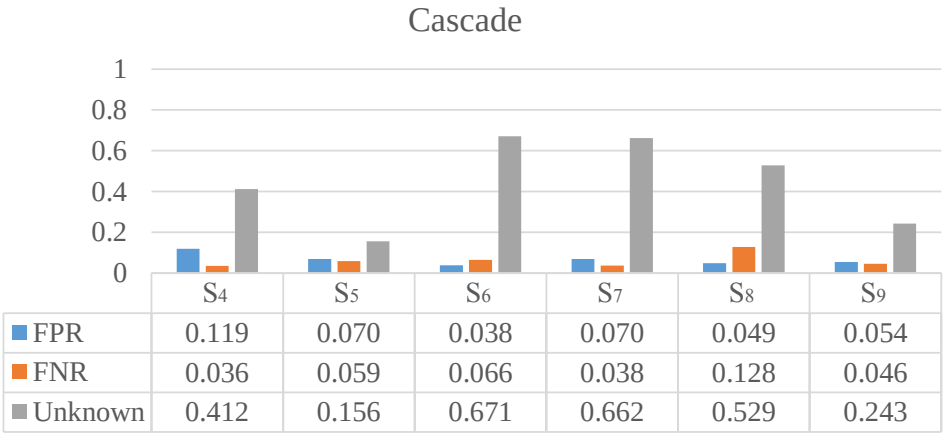
(c)

Figure 3.15: Detection of unknown attacks that target different IC components are evaluated for various models, including (a) a decision tree, (b) the cascade model, and (c) an RNN.



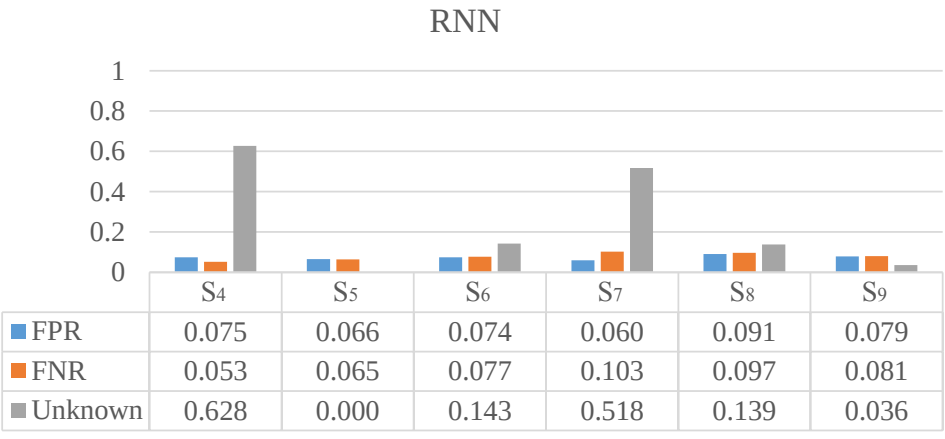
	S_4	S_5	S_6	S_7	S_8	S_9
FPR	0.067	0.056	0.056	0.038	0.035	0.061
FNR	0.133	0.130	0.127	0.058	0.251	0.117
Unknown	0.471	0.679	0.623	0.640	0.876	0.650

(a)



	S_4	S_5	S_6	S_7	S_8	S_9
FPR	0.119	0.070	0.038	0.070	0.049	0.054
FNR	0.036	0.059	0.066	0.038	0.128	0.046
Unknown	0.412	0.156	0.671	0.662	0.529	0.243

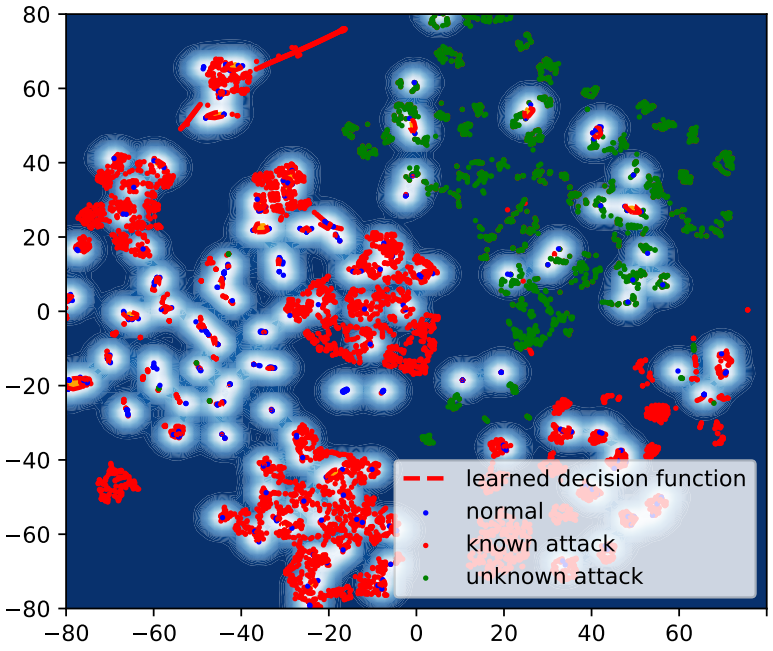
(b)



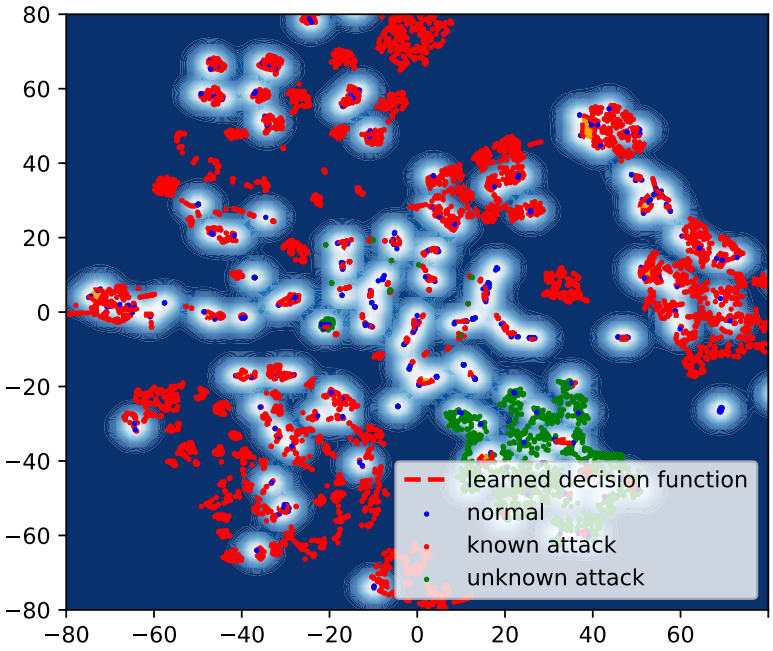
	S_4	S_5	S_6	S_7	S_8	S_9
FPR	0.075	0.066	0.074	0.060	0.091	0.079
FNR	0.053	0.065	0.077	0.103	0.097	0.081
Unknown	0.628	0.000	0.143	0.518	0.139	0.036

(c)

Figure 3.16: Detection of unknown attacks that exploit different strategies (listed in Table 2.2) are evaluated for various models, including (a) a decision tree, (b) the cascade model, and (c) an RNN.



(a)



(b)

Figure 3.17: Legitimate JTAG operation and attacks for the OpenSPARC T2 are plotted using t-SNE. Two types of attacks (targeting an IC component not included in training) are regarded as unknown, one located in (a) an open space, and the other located in (b) an area overlapping with legitimate operation.

identify all types of attacks not included in training. Collecting JTAG attacks is an important but difficult process because 1) hardware attackers typically do not publish their attacks, and 2) unknown attacks always exist. Nevertheless, hardware companies should still be aware of potential attacks, not only for mitigating the current vulnerability, but also for improving security of future designs. We suggest that hardware companies build a database of JTAG attacks and collect potential attacks using similar strategies as anti-virus software, including for example analyzing user reports and periodic security examination. When new attacks are collected, it needs to be checked if they can be effectively detected by the existing detector with high confidence. If not, the detector should be re-trained and uploaded to the on-chip memory.

A false positive occurs when a legitimate user is classified as an attacker. It causes inconvenience for legitimate users because the JTAG will be permanently protected in the case of false positives. There exists a tradeoff between false positive rate (FPR) and false negative rate (FNR), meaning that a reduction of the FPR leads to a higher probability that an attacker escapes detection. The evidence collection described in Section 3.3 is an effective method for mitigating both false positives and false negatives. According to the results in Figure 3.13, an RNN can reduce FPR and FNR to 1.3% and 1.2%, respectively, for the OpenSPARC T1. Considering that an FPR of 1.2% is small, and that in-field uses of JTAG involve only a few scenarios (e.g., flash programming and software debugging) and are typically not frequent [110, 111], the JTAG is not likely to enter the protection mode. In other words, if used legitimately, the JTAG is expected to be accessible for a sufficiently long period.

A more holistic way of reducing false alerts is to pass the prediction to the system, such that the final decision is made globally by the system instead of the proposed detector. The system may need to monitor features from other modules/interfaces and exploit more complex classification models, in order to make more accurate decisions concerning access to the JTAG. This topic, however, is beyond the scope of this dissertation and is suggested as a topic to be explored in future work.

3.8 Summary

This chapter discusses how the characterized features and collected data can be analyzed for detecting illegitimate JTAG accesses. To capture existing serial dependence in JTAG operation, several detection models have been developed and evaluated, including sliding-window-based detectors, sequence models, and a representative-based anomaly detector. In addition, to detect unseen attacks (or to minimize open-space risk), a cascade classifier, combining a one-class SVM and a binary SVM, is developed. Experiments using the OpenSPARC T1, T2 and the LEON3 processors demonstrate effectiveness of the detection models. Several ML models, including a decision tree, a random forest, SVMs and k-NN, show similar performance. The experiments also demonstrate that an RNN is more effective in detecting attacks and even unseen attacks compared to the sliding-window-based detectors and the cascade model. For model selection, a designer needs to consider the trade-off between detection performance and overhead. On-chip overhead for attack detection is discussed next in Chapter 5.

Chapter 4

Detection of IJTAG Attacks

The IEEE Standard 1687 (IJTAG), an extension to the IEEE Standard 1149.1, facilitates efficient access to embedded instruments by supporting reconfigurable scan networks. Specifically, IJTAG allows each IP to be wrapped by a test data register (TDR) whose access is controlled by a segment insertion bit (SIB) or a scan-mux control bit (SCB). Because the TDRs and the SIB/SCB network are typically not made known to the public but are critical for accessing embedded instruments[1], they might be used for illegitimate purposes. Machine learning has been proposed to detect attacks to the JTAG (in Chapter 3), but the large number of instruments and parallel execution enabled by the IJTAG produce high-dimensional data, which poses a challenge to on-chip detection. In this chapter, we propose to compress the high-dimensional data collected from IJTAG operation based on a low-density parity-check (LDPC) matrix. Compression is possible because IJATG operation is likely to result in a sparse data set. According to the experiments using a modified version of the OpenSPARC T2, the use of LDPC matrices exhibit a good trade-off between detection accuracy, compression and hardware overhead.

The rest of this chapter is organized as follows. Section 4.1 reviews the background of IJTAG and illegitimate IJTAG accesses, and studies existing work on IJTAG protection. Section 4.2 elaborates upon the LDPC-based feature reduction and the process of IJTAG attack detection. Sec-

[1] An instrument refers to an IP core with an IJTAG-compliant interface.

tion 4.3 describes changes to the OpenSPARC T2 in order to incorporate IJTAG, and evaluates the performance of feature reduction. Section 4.4 discusses several issues concerning the proposed approach, and Section 4.5 summarizes the work of this chapter.

4.1 Background

The increasing complexity of ICs requires integration of a large number and variety of IPs. Learning the setup, integration, and test procedure of each IP imposes additional burden on IC designers and test engineers. To mitigate the resulting test issues, IEEE Std 1687 (IJTAG), as an extension to the IEEE Std 1149.1, has been developed to facilitate efficient access to embedded instruments [33]. IJTAG allows scan chains in each instrument to be configured using a segment insertion bit (SIB) or a scan-mux control bit (SCB). In addition to SIBs/SCBs, IJTAG also defines a standard interface for accessing an instrument, an instrument connectivity language (ICL), and a procedure description language (PDL). The adoption of IJTAG not only reduces the effort in learning the setup, integration, and test procedure, but also allows IPs to be modified or added locally without affecting other instruments.

4.1.1 IJTAG Architecture

In an IJTAG network, access to an instrument is configured using SIBs/SCBs, whose structure is shown in Figure 4.1. A SIB gates an instrument, such that access to the instrument is possible only if the SIB is open (i.e., the signal "Select" is high); otherwise, the instrument is bypassed. An SCB controls a multiplexer that exclusively selects a scan chain. A SIB or an SCB consists of two flip-flops (FFs), for shifting and updating. Figure 4.2 shows an example of an IJTAG architecture with four instruments. To access the TDR of an instrument, the user needs to operate the TAP controller to shift in a logic one to its gating SIB, and/or shift in a proper value to its selecting SCB. After the UPDATE-DR state loads the values into the updating FF of the SIBs/SCBs, the target TDR

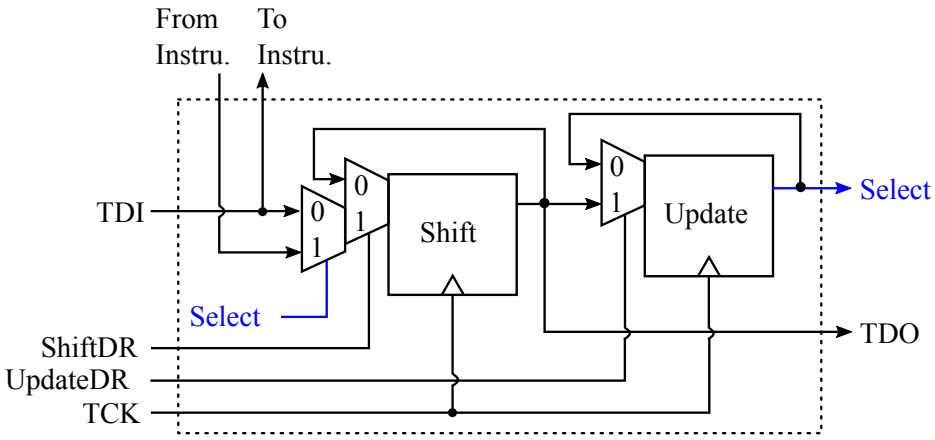


Figure 4.1: The structure of a segment insertion bit (SIB).

is configured as a part of the chain. For example, to access the TDR of MEM1 in Figure 4.2, the user needs to operate the TAP controller and supply the proper opcode into the instruction register (IR) that selects the SIB/SCB network configuring MEM1. Now the TAP controller has access to a four-bit chain, i.e., SIB1→SCB2→SIB3→SIB4. Next, the user needs to shift in '1100' during the SHIFT-DR state (the LSB is shifted in first) and update the value to the SIBs/SCBs during the UPDATE-DR state. The TDR of MEM1 is from hereon configured within the scan chain, meaning that the TAP controller can now access it. Note that there might exist a hierarchy of SIBs/SCBs, in which case, accessing a TDR involves asserting multiple levels of SIBs/SCBs.

A TDR, consisting of a shift register, can be accessed via a standard eight-port interface, as shown in Figure 4.3. A serial data stream is shifted into the instrument via the TDR_SIN port and shifted out via the TDR_SOUT port. The other ports are controlling signals decoded from the TAP controller. The TDR can supply test vectors or debugging commands to the instrument when the TDR_UPDATE is set, and capture test response data when the TDR_CAPTURE is set. Figure 4.3 shows an industrial example of a memory wrapped by a six-bit TDR [112]. Bit one is used for resetting the BIST (built-in self-test). Bit two is used for starting the BIST and capturing the status of the BIST (busy or not). Bits three and four are used for updating a new BIST mode and capturing the existing BIST mode. Bit five only has a shift cell since it can only be used for capturing the BIST result (pass or not), but not for updating. Bit six is used like a SIB/SCB, because it can

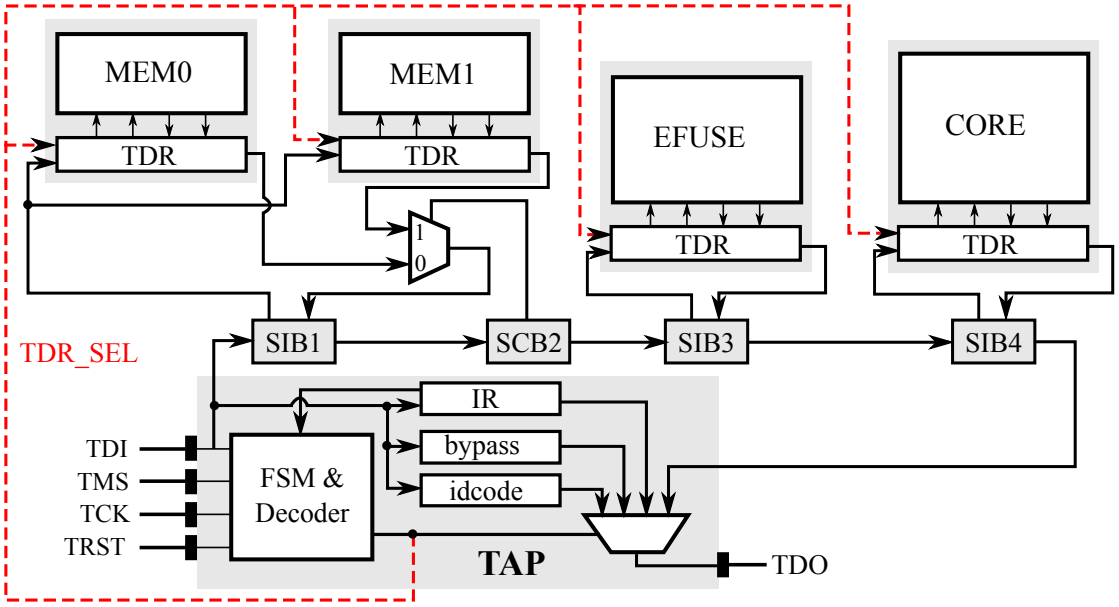


Figure 4.2: An example of an IJTAG architecture composed of four instruments, each of which is gated by a SIB and/or selected by an SCB.

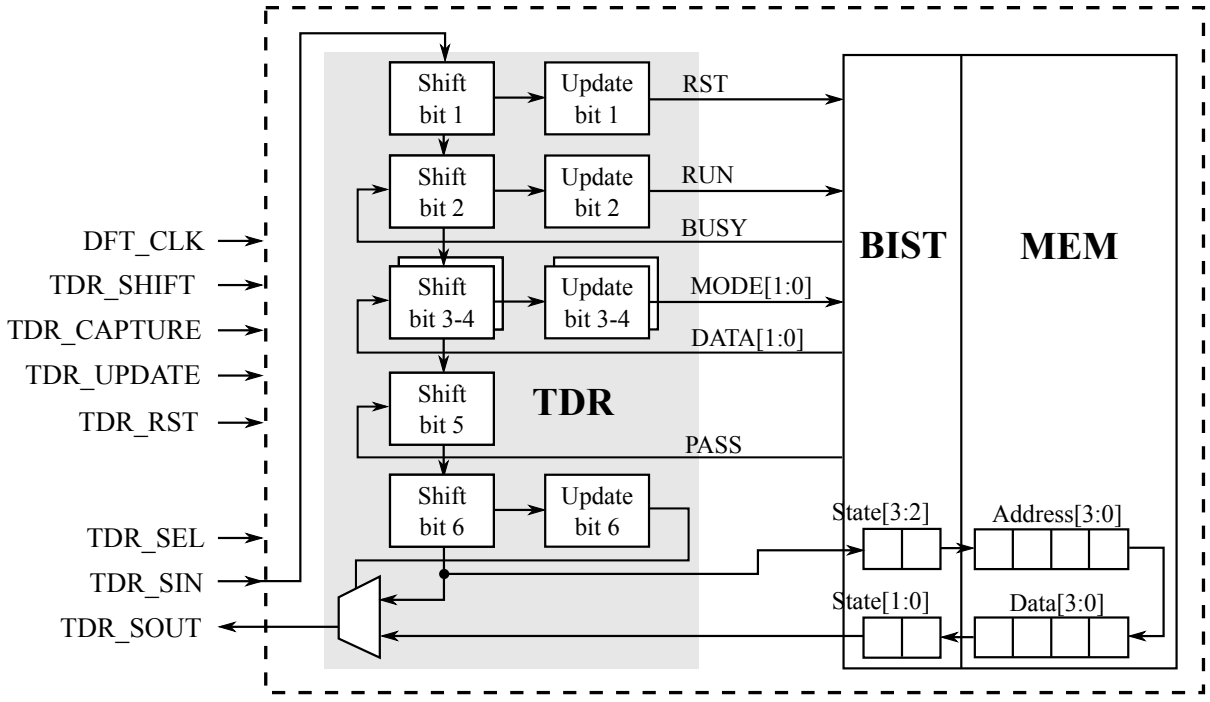


Figure 4.3: An industrial memory macro wrapped by a six-bit TDR.

be used for accessing the next level of registers. Since these bits can be set simultaneously, their corresponding operations can also be executed in parallel. For example, a simultaneous setting of

bits three to five indicates the parallel execution of selecting the BIST mode and querying the BIST result.

4.1.2 Prior Work

Various countermeasures have been proposed to protect the IJTAG, including access restriction, encryption, attack detection and obfuscation. Access to the IJTAG can be restricted through fusing off the JTAG port, or monitoring if a user attempts to assert the SIBs that gate secure scan chains [113]. Both methods disable access to entire or parts of a scan chain, and therefore hinder in-field debugging and programming. The work in [114] proposes to insert key bits into scan chains such that the SIBs are opened only if a correct key is supplied. The use of an LFSR [115] and honeytraps [116] makes the key more complex. A honeytrap refers to the SIBs that are used as key bits for other SIBs. This makes key searching even more complex than the case where key bits are located in scan chains. However, the key might still be leaked during distribution [117]. The work in [118] improves the security of the IJTAG using a challenge-response protocol, which, however, requires availability and security of network communication. Another countermeasure involves detection of illegitimate IJTAG access through checking if the number of shifting cycles exceeds a pre-defined range [58]. This simple, rigid rule, however, is not able to detect complex attacks, and may also result in many false positives [119, 120].

Although machine learning proves to be an effective approach for detecting JTAG attacks [119, 120], detection of illegitimate IJTAG access is more challenging than JTAG, primarily for two reasons. First, characterizing IJTAG operation may require many more features. Different from JTAG operation that can be characterized using the sequence of instruction-register opcodes [120], IJTAG operation involves configuring a hierarchy of SIBs/SCBs and setting/resetting the bits in a large number of TDRs. Second, the IJTAG not only allows simultaneous assertion of multiple SIBs/SCBs, but also allows simultaneous setting of multiple bits in a TDR, implying that many more combinations of operations become possible compared to JTAG.

4.2 IJTAG Attack Detection

This section describes the features used for characterizing IJTAG operation, reduction of the features, and attack detection using a random forest classifier.

4.2.1 IJTAG Attacks

IJTAG-compliant instruments may also be accessed illegitimately because of their accessibility via the JTAG port. For example, prior work has demonstrated that cryptographic keys can be derived by analyzing the data dumped from scan chains [8]. An attacker may also reverse engineer the SIB/SCB network and the meaning of each bit in TDRs, which can then be used to derive data from on-chip memory [25], update firmware [26], and control chip operation [27]. Because IJTAG has gained growing support from EDA vendors, and is projected to be adopted widely in industry [34], the security of IJTAG is therefore a topic of importance.

As illustrated by Figure 1.2, an attacker is assumed to have access to only the JTAG port and is, at least initially, unaware of which private JTAG functions, instructions, and data registers are implemented. Thus, the attack strategies described in Table 2.1 can still be employed. Further, it is assumed that the attacker does not know the SIB/SCB network and how to operate the TDR bits within each instrument. To uncover the SIB/SCB network, the attacker would tentatively load one and zero to each bit, and observe if the length of the chain between the TDI and TDO changes. The attacker can repeat this interrogation until the whole SIB/SCB network is uncovered. Once gaining access to a TDR, the attacker can uncover the operation corresponding to each TDR bit, exploiting strategies similar to those described in [119]. More precisely, an attacker can set each TDR bit and check the response. The attacker can also vary the order of bit setting, and examine possible interactions between them. Further, since the attacker can set multiple bits simultaneously, other than sequentially, for checking their interaction, reverse engineering an IJTAG network becomes even more efficient.

Table 4.1: FEATURES USED FOR CHARACTERIZING IJTAG OPERATION.

Category	Index	Description	Count
SIBs/SCBs	F1	SIB/SCB bits	N_S
	F2	No. of asserted SIBs/SCBs	1
	F3	No. of bit transitions[2] in SIBs/SCBs	1
TDRs	F4	No. of ones in each TDR	N_T
	F5	No. of bit transitions in each TDR	N_T
	F6	No. of dependent bits in TDRs	1
TAP	F7	No. of TMS transitions	1
	F8	TEST-LOGIC-RESET activated?	1

4.2.2 Feature Extraction

As described in Section 4.1.1, IJTAG allows simultaneous setting of multiple bits in a TDR, which means IJTAG operations can be performed in parallel. This typically happens in two scenarios, namely independent operation and pipelining operation. TDR bits are dependent if swapping their order leads to a different result. For example, bit two and bits three to four in Figure 4.3 are dependent because a different BIST mode (operated by bits three to four) may initiate a different BIST process (operated by setting bit two). Dependent bits are typically not set simultaneously unless they are operated in a pipelining manner. For example, bit two and bits three to four, if set simultaneously for a pipelining use, means that the BIST mode is set for the next BIST process rather than the current one. Besides dependent bits, TDR bits may even conflict if setting them simultaneously causes uncertain results. For example, bit one, used for resetting the BIST, conflicts with bit two, which is used for starting the BIST. Although this conflict can be handled through a careful design, simultaneously initiating operation should be avoided. In other words, if conflicting operations are detected, operation should be immediately labeled as an attack.

IJTAG operation, consisting of SIBs/SCBs, TDRs and the TAP controller, is characterized using the features shown in Table 4.1 (N_S and N_T refer to the number of SIBs/SCBs and the number

[2]A bit transition refers to a SIB/SCB changing from asserted to de-asserted, or from de-asserted to asserted.

of TDRs, respectively). F1 refers to the bits stored in the SIBs. F2 and F3 characterize the number of ones (i.e., the number of opened SIBs) and the number of SIB transitions compared to the previous cycle (i.e., the number of SIBs that change from open to closed, or from closed to open). Similar to F2 and F3, F4 and F5 are used for characterizing the bits in each TDR. F6 represents the number of dependent bits found in all TDRs. F7 and F8 characterize the operation of the TAP controller, namely the number of TMS transitions (from low to high, or from high to low) and whether a TEST-LOGIC-RESET state is encountered.

Recall that features characterizing JTAG operation, as described in Section 2.3, are collected during the UPDATE-IR state. For the IJTAG, in order to capture the change of values in SIBs/SCBs and TDRs, features are collected during the UPDATE-DR state instead. Correspondingly, an operation cycle is defined as the period between two successive UPDATE-DR states. To better reflect the sequential feature of IJTAG operation, a sequence of operation cycles, rather than a single one, are monitored. Specifically, data are collected using a sliding window (with w operation cycles) in an overlapping manner.

4.2.3 Compressed Sensing

The data collected using a sliding window typically have a large dimension, but they are likely sparse. This is because the operations corresponding to the TDR bits, in most cases, should still be executed in specific orders. A simultaneous setting of all TDR bits (or most of them) is not practical. This observation means that the dimensionality of the data is likely reducible. A possible technique to reduce dimension is fixed-length encoding (i.e., encode each possible combination of the observed data using a fixed-length code, whose length depends on the number of combinations in the observed data). However, fixed-length encoding may not reduce the dimension effectively because most combinations, although absent from the training set, are still possible. Another technique involves compressed sensing that aims to compress and reconstruct high-dimensional data with low complexity [121]. Let $x \in \mathbb{R}^n$, $z \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$ ($m \ll n$), the compression and

reconstruction of x can be formulated as

$$z = Ax \tag{4.1}$$

and

$$\hat{x}_{Lasso} := \underset{x}{\operatorname{argmin}} \|z - Ax\|_2^2 + \lambda_1 \|x\|_1 \tag{4.2}$$

respectively. The compression is simply a multiplication of x and a matrix A representing a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$. The reconstruction involves a linear regression using Lasso regularization (i.e., $\|x\|_1$). Note that although reconstruction is not necessary for classification, it is still useful because it evaluates the quality of the compression. The linear regression formulated in Equation 4.2 is underdetermined ($m \ll n$), meaning that the solution of $\hat{x}$ is not unique. To achieve a sparse solution of $\hat{x}$, the Lasso regularization, rather than $\|x\|_2^2$, is used. The performance of the Lasso declines if the columns of A are highly correlated (in this case, the Lasso simply chooses one column of A). To mitigate this problem, matrix A should satisfy the restricted isometry property (RIP) that requires a matrix to be "almost" orthonormal, at least when operating upon sparse vectors. However, there exist no effective approaches to construct such matrices, although some matrices, like Gaussian matrices, satisfy the RIP with exponentially high probability [122].

As studied in [123], LDPC codes, originally used as error correcting codes, show an outstanding performance when used for compressed sensing. An LDPC code can be represented using a binary matrix (typically called the LDPC matrix) or a bipartite graph (which is also referred to as a Tanner graph) [124]. Figure 4.4(a) shows an example of a Tanner graph that represents the same LDPC code as the matrix A in Equation 4.3. The graph consists of n variable nodes (the number of bits in a code word) and m check nodes (the number of parity bits). Check node c_i is connected to variable node v_j if the element a_{ij} of A is a 1. The Tanner graph shown in Figure 4.4(a) can also be represented using a tree that is constructed by traversing the adjacent nodes non-repeatedly as shown in Figure 4.4(b). Note that, starting from the root node (i.e., v_0), there exist many cyclic paths, among which the length of the shortest path is defined as local girth, g. A large value for g means that the root node is significantly independent from other variable nodes, and therefore is

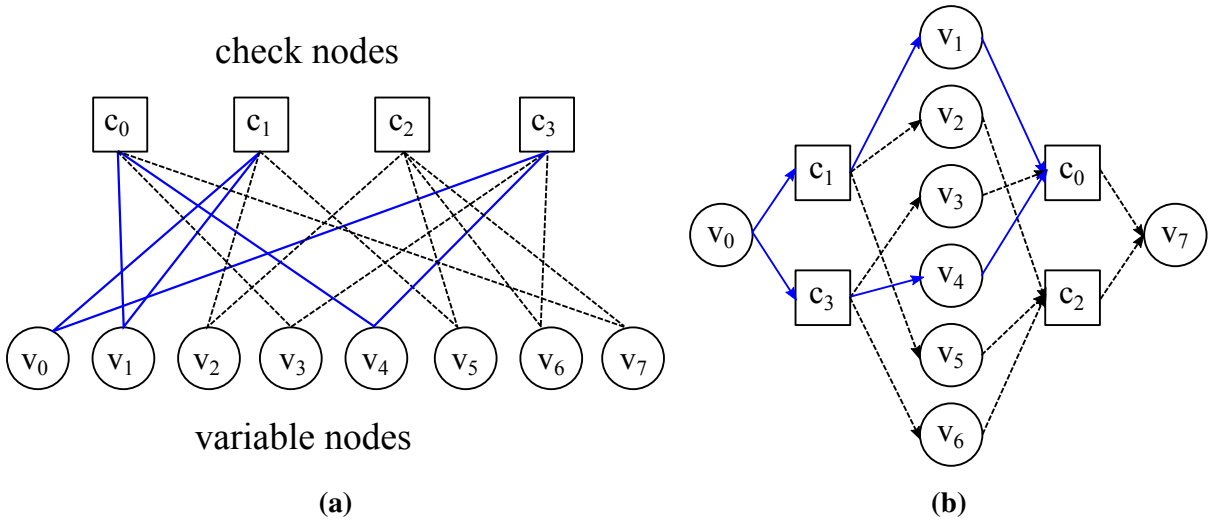


Figure 4.4: (a) An example of a Tanner graph with eight variable nodes and four check nodes. (b) The adjacent variable nodes and check nodes of the Tanner graph in Figure 4.4(a) are traversed starting from v_0.

more likely to lead to an orthonormal matrix. In addition, because the entries of an LDPC matrix are either one or zero, feature reduction using an LDPC matrix involves only additions, rather than matrix multiplication.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{4.3}$$

To construct A, two variables, namely the reduced dimension (m) and the number of ones in each column of A (d), need to be determined. Because the complexity of the original data is reflected by its density (number of non-zero entries), the optimal value for m might also be close to the average density of the original data. The number of ones in each column of A (d), reflecting the density of A, affects the orthogonality of A. According to the analysis in [123], as d increases (g decreases as a result), the correlation between the columns of A decreases when $g > 4$, and increases again

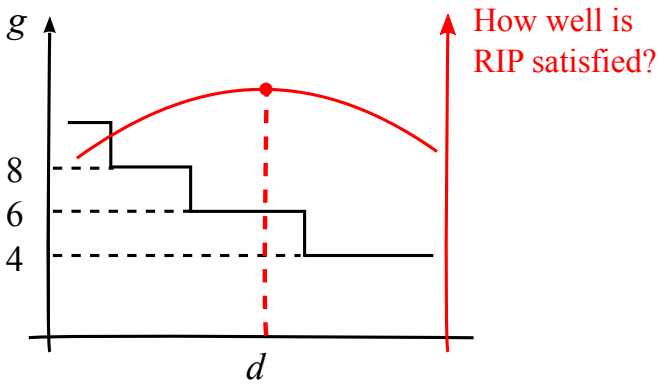


Figure 4.5: As d (density of A) increases, g (the local girth) decreases. The largest d that satisfies $g > 4$ is chosen for constructing the LDPC matrix A.

when $g = 4$. Thus, the largest d that satisfies $g > 4$ is chosen, as shown in Figure 4.5 (note that this value should be close to but may not be the theoretically-optimal value).

4.2.4 Overall Flow

Figure 4.6 shows the overall flow of IJTAG attack detection. During an UPDATE-DR state, the bits in each SIB/SCB and each TDR are collected for primary checks, i.e., the user is labeled as an attacker immediately if an illegal opcode (not correspond to any JTAG function) is loaded into the IR or conflicting bits in TDRs are detected. If these checks do not detect an attack, then the features described in Table 4.1 are collected. The dimensionality of the collected data is

$$s = (N_S + 2 + 2N_T) \cdot w + 3 \cdot w \tag{4.4}$$

The first term in equation (4.4) corresponds to features F1-F5 (as shown in Table 4.1) that can be reduced using an LDPC matrix due to their sparsity. The second term corresponds to features F6-F8 whose dimension can be reduced by deriving their statistics (such as max, min, and/or mean) within a window. Thus, the reduced dimensionality is

$$s' = m \cdot w + 3 \cdot r \qquad (1 \leq r \leq 3) \tag{4.5}$$

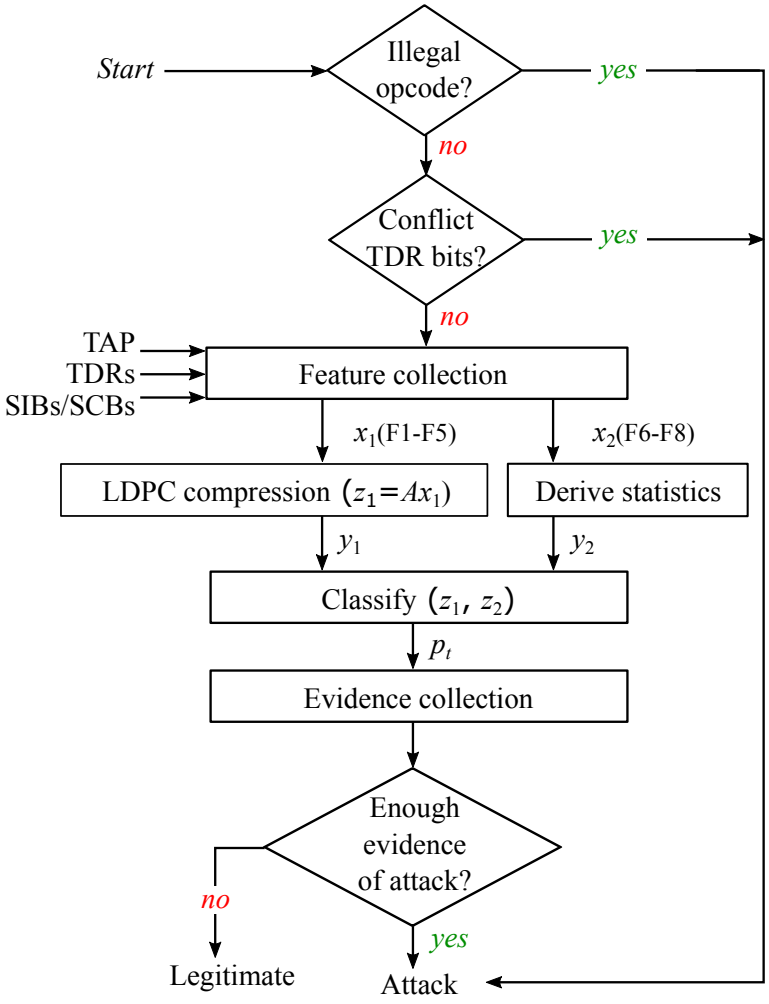


Figure 4.6: The overall flow of IJTAG attack detection.

where r is a constant between 1 and 3 because F6-F8 may only use partial, but not all, the statistics of max, min, and mean. According to equation (4.5), the dimensionality of the reduced data is bounded because neither term depends on the number of SIBs/SCBs and TDRs. The reduced data are supplied to a pre-trained random forest classifier that labels IJTAG operation as either legitimate or as an attack. A random forest, consisting of an ensemble of decision trees, is a preferred model for classifying high-dimensional data [66]. Finally, to avoid false alerts caused by the variance within IJTAG operation, the evidence collector described in Section 3.3 is employed.

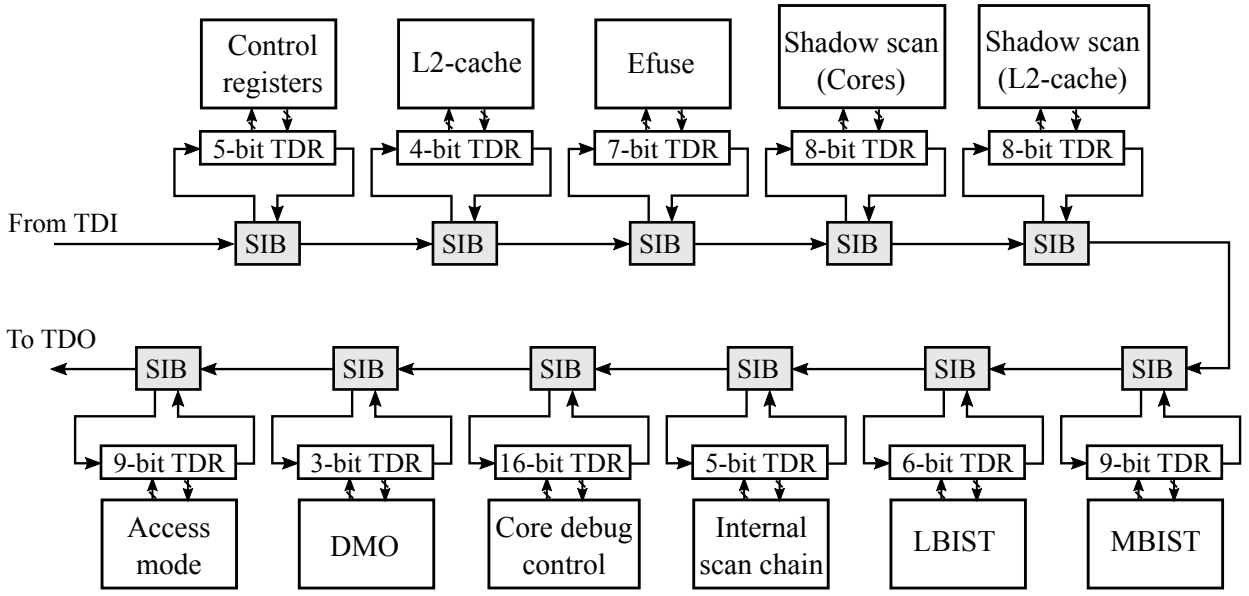


Figure 4.7: The OpenSPARC T2 is partitioned into 11 sub-systems, and then each sub-system is wrapped by a TDR. In addition, each TDR is gated by a SIB, and all SIBs comprise a daisy chain that is accessible via the JTAG port.

4.3 Experiments

This section evaluates the performance of IJTAG attack detection and the LDPC-based compressed sensing. The evaluation is based on a modified version of the OpenSPARC T2.

4.3.1 Modification of OpenSPARC T2

The OpenSPARC T2 processor is used as the platform for experiments[3]. The JTAG of the OpenSPARC T2 not only has 32 scan chains, but also can be used for dozens of testing/debugging functions. Based on these functions, the OpenSPARC T2 is partitioned into 11 sub-systems as exhibited in Figure 4.7, and then each sub-system is wrapped by a TDR by learning from industrial examples [112]. Then, a set of SIBs are inserted such that each wrapped sub-system is gated by a SIB. All SIBs comprise a daisy chain that is accessible via the JTAG port.

[3]Nevertheless, the proposed detector is generic, and can be applied to other IJTAG architectures.

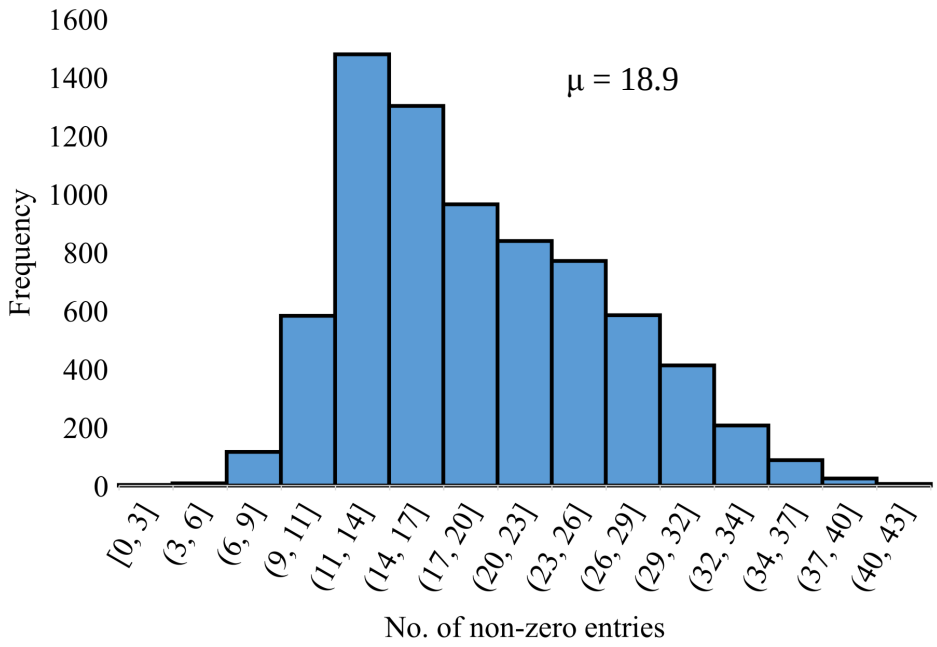


Figure 4.8: The density distribution (i.e., number of non-zero entries) for the collected data whose dimension is 280.

To operate the T2 IJTAG, a set of legitimate operations (135 traces) are created based on the documentation of the OpenSPARC T2. Each trace achieves a basic operation, like reading specific cache lines. Independent operation and pipelining operation allowed by the IJTAG are also exploited when creating the traces. The number of TDR bits that are operated simultaneously (which is equivalent to the degree of parallelism) varies from two to four, depending on specific scenarios. In addition to legitimate IJTAG operation, a variety of attacks (156 traces) are also created, based on the strategies described in Section 4.2.1, namely uncovering the SIB network and the meaning of TDR bits. Note that parallel execution can also be exploited by an attacker, but the degree of parallelism may vary in a much larger range assuming that the attacker searches many more combinations of the TDR bits. The resulting traces are then simulated using the modified OpenSPARC T2 and the features (as described in Table 4.1) are collected using a sliding window of eight cycles. The density of the collected data, as shown in Figure 4.8, reveals that most data have fewer than 40 non-zero entries (the dimension of the data corresponding to F1-F5 is 280). This also verifies the initial assumption that data collected from IJTAG operation are sparse.

4.3.2 Evaluation of LDPC Matrices

The collected data corresponding to F1-F5 are then compressed using an LDPC matrix. To construct an LDPC matrix A, two variables, namely m (the reduced dimension) and d (the number of ones in each column of A), must be determined. According to the analysis in Section 4.2.3, the largest d that satisfies $g > 4$ is preferred, and this value, through calculation, is three (assuming $m = 30$). This value is verified through simulation. Specifically, the collected data, both legitimate and attack, are compressed and reconstructed using LDPC matrices with different values of d, and then the mean squared error (MSE) of the reconstructed data is evaluated. The simulation result, as shown in Figure 4.9, verifies the calculated value of m (i.e., 3), and demonstrates that the quality of compression becomes worse if d is too large or too small. Figure 4.9 also compares LDPC matrices to three baseline matrices, namely random matrices with degree $= d$ (each column has d ones whose positions are random), arbitrarily-random matrices (each column has arbitrary number of ones) and Gaussian matrices (each entry is a Gaussian i.i.d random variable). According to the comparison, an LDPC matrix outperforms the other matrices when $d = 3$ but not for other values. Note that a random matrix with degree $= d$, showing a convergence when $d > 5$, also achieves competitive MSE, which however is still inferior to an LDPC matrix with $d = 3$.

4.3.3 Evaluation of IJTAG Attack Detection

After compressing the data corresponding to F1-F5 and deriving the statistics from the data corresponding to F6-F8 (only max is used in this experiment), the data is supplied to a random forest with three trees for evaluation. The evaluation uses a five-fold cross-validation. The LDPC-based feature reduction is compared with a baseline approach, namely feature selection using a decision tree. More precisely, the features that demonstrate superior capability of reducing the impurity of the data are selected. The performance of both approaches is measured by error rate, false positive rate (FPR), and false negative rate (FNR), as shown in Figure 4.10. According to the results in Figure 4.10(a), the features reduced by an LDPC matrix demonstrate similar error rate

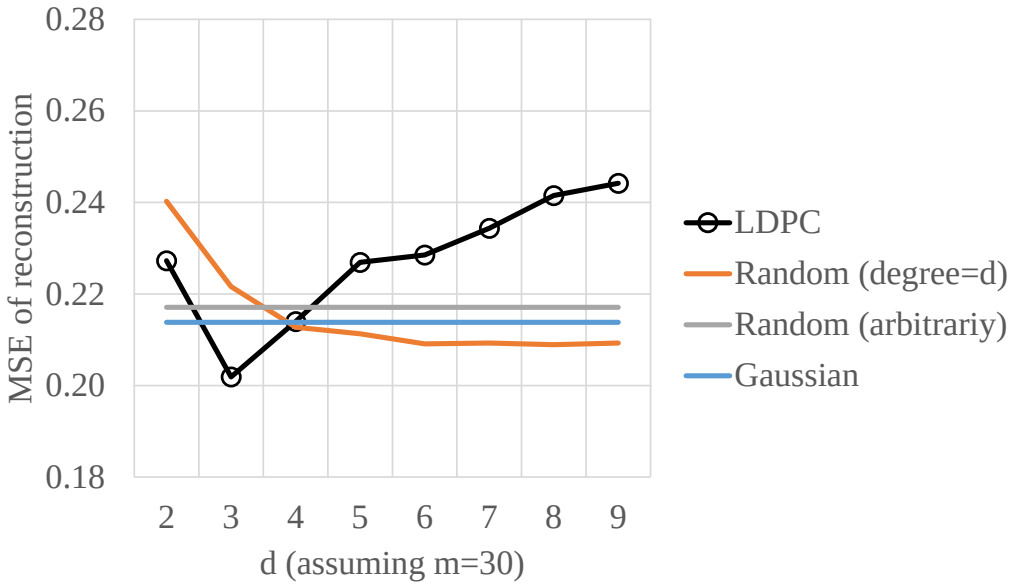
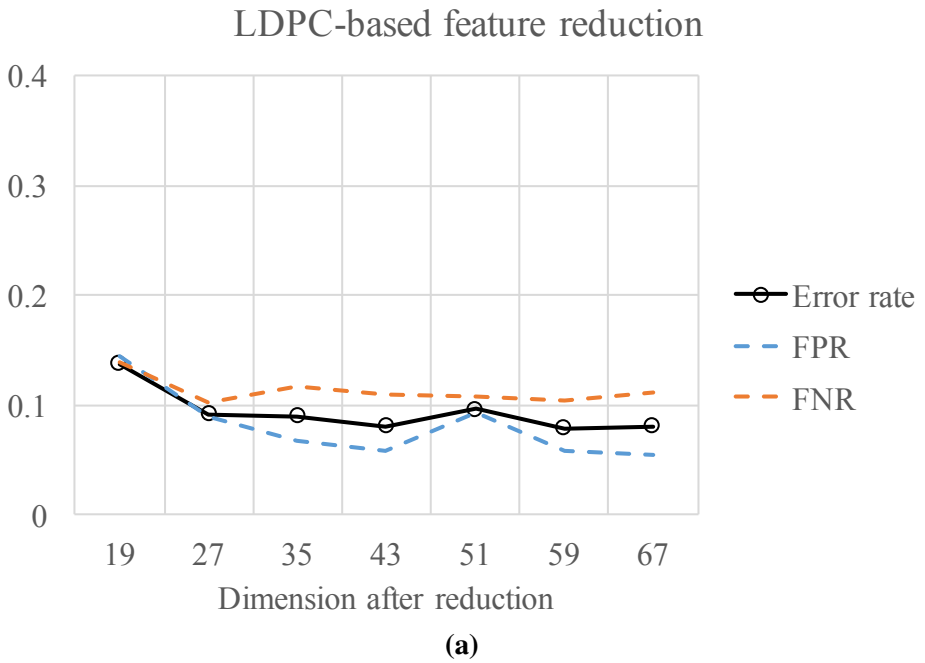


Figure 4.9: The performance of reconstructing the original data is evaluated for LDPC matrices and three other types of matrices.

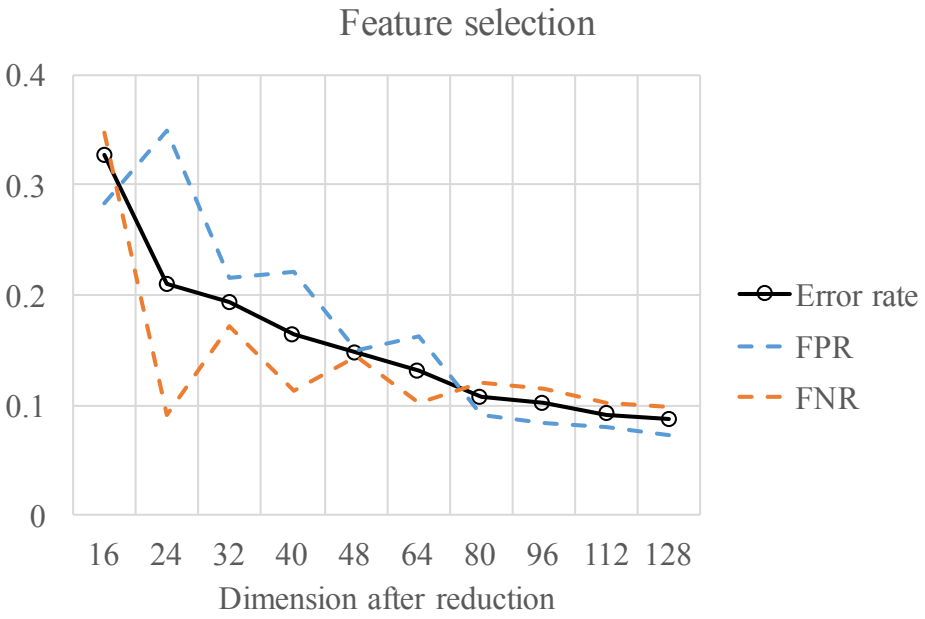
as the original features (0.084), for a reduced dimension larger than 27. Let the reduced dimension be 27 (meaning that $m \cdot w = 24$), the LDPC matrix reduces the original dimension (304) by 91% without weakening the performance of classification. It is worth noting that $m \cdot w = 24$ is somewhat larger than the average density of the original data (as shown in Figure 4.8). The feature selection, as shown in Figure 4.10(b), is also effective for dimension reduction, but is less efficient. In other words, to achieve a similar level of error rate, the reduced dimension needs to be higher than 100, which is almost four times larger than the LDPC-based feature reduction.

4.4 Discussion

Several aspects concerning the proposed detector still need further examinations. First, although the modified OpenSPARC T2 only has 11 instruments, the LDPC-based feature reduction is effective for a system with more instruments. This is because the reduced dimension depends more on the sparsity of the data than the number of instruments. However, more instruments may



(a)



(b)

Figure 4.10: The data, whose dimension is reduced by (a) the LDPC-based feature reduction and (b) feature selection using a decision tree, are classified using a random forest.

incur wiring overhead since the SIB/TDR values need to be tapped for a global detector. This important aspect is a topic that should be explored in future work (also presented in Chapter 6).

TABLE 4.2: SYNTHESIS RESULTS ARE COMPARED FOR THE RANDOM FOREST (WITH THREE TREES) DETECTORS (GATE EQUIVALENT = TOTAL AREA / AREA OF A TWO-INPUT NAND GATE).

	Original detector	w/ feature selection	w/ LDPC
Data dimension	304	96	27
Area (gate equivalent)	20,585	14,384	11,749
Compare to area of JTAG/IJTAG	14%	9.8%	8%
Latency (clock cycles)	4	4	8

Second, a comprehensive security analysis should take into consideration all types of attacks, which is difficult since different types of attacks will likely arise. Nevertheless, the experiments in Section 3.6 show that a ML-based detector has potential to detect unseen attacks.

Third, the LDPC-based feature reduction, although requiring additional adders, reduces the size of registers storing features and the depth of each decision tree. Table 4.2 shows the synthesis results using Synopsys Design Compiler with a 0.18 μm library. According to the results, a random forest detector with LDPC-based feature reduction adds only 8% chip area compared to the JTAG and IJTAG, which is smaller than the technique using feature selection. As for latency, each row of the LDPC matrix has $\frac{n \cdot d}{m}$ ones on average, meaning that $\frac{n \cdot d}{m}$ features, on average, need to be added. This consumes $\log_4 \lceil \frac{n \cdot d}{m} \rceil$ clock cycles assuming that the adder in each clock cycle has at most four inputs. Since the LDPC matrix has m rows, the additions described above need to be operated for m times. This results in $\log_4 \lceil \frac{n \cdot d}{m} \rceil + m - 1$ clock cycles in total if the m additions are executed in a pipeline manner.

4.5 Summary

In this chapter, an ML-based detector and LDPC-based feature reduction are developed for detecting real-time attacks of IJTAG-compliant systems. According to experiments based on a

TABLE 4.3: THE PROPOSED APPROACH IS COMPARED TO PRIOR WORK.

Technique	Security	Overhead	Drawback
Access restriction [113]	+++	++	Hinder in-field test/debug
SIB locking [114–116]	++	+	Key leakage during distribution or via power analysis
Challenge-response [118]	+++	++	Require availability and security of a network
ML-based attack detection	++	++	False positives and false negatives

modified version of the OpenSPARC T2, the on-chip detector only adds $\sim$ 8% more chip area to the IJTAG. Further, the use of an LDPC-based feature reduction eliminates 91% of the features, and reduces circuit size by 43% without affecting detection accuracy. Nevertheless, it is hard to compare the proposed detector to other techniques shown in Table 4.3 because there is no universal metric to evaluate the level of security each method provides. However, these techniques might be combined to achieve complementing protection to the IJTAG.

Chapter 5

JTAG Restriction and Detector Implementation

The detector needs to be implemented on chip using either hardware or software, in order to monitor real-time JTAG/IJTAG operation and detect attacks in time. The study of typical JTAG/IJTAG operations demonstrates that attacks can take as few as dozens of clock cycles. Thus, the classification also needs to be completed within this period. Upon detection of an attack, access to the JTAG needs to be restricted immediately, in order to prevent further attacks (e.g., leakage of sensitive on-chip information). In Section 5.1, restriction of JTAG access is described, which not only prohibits observability and controllability provided by the JTAG, but also hinders adversarial attack of the ML model. An adversarial attack involves reverse engineering of the ML model (i.e., uncovering structure and parameters of the ML model) or simply searching for attacks that can escape detection. Section 5.2 analyzes security provided by the access restriction and ML-based detection. Section 5.3 describes implementation of the sliding-window-based detectors, and then evaluates the resulting overhead. Section 5.4 discusses several issues concerning the implementation, and finally, Section 5.5 summarizes the chapter.

TABLE 5.1: FOR EACH TYPE OF DR, CONTROLLABILITY AND OBSERVABILITY ARE PROHIBITED THROUGH SPECIFIC ACTIONS.

Type of DR	Disable controllability	Disable observability
Capture-only	N/A	Read from decoy shadow register
Update-only	Disable update operation	N/A
Capture/update	Only write to decoy shadow register	Read from decoy shadow register
Scan-accessible	Disable shift operation	Shift out LFSR for cycles equal to DR length, then shift out decoy DR data

5.1 Restriction of JTAG Access

Upon detection of an attack, the JTAG needs to be protected, to prevent further attacks. However, if the attacker knows which operation triggers the protection, then he/she may avoid that operation in future attacks (to other chips). Consequently, that particular triggering operation should be obfuscated. In order to achieve this objective, access to DRs is modified as shown in Figure 5.1. It prevents the chip from being controlled/modified, and also prevents on-chip data from being observed.

Data registers (DRs) are categorized into four types, namely capture-only DR, update-only DR, capture/update DR, and scan-accessible DR, as shown in Figure 5.1. It should be noted that some DRs may have multiple types. For example, it is not unusual for a DR to have capture, update and scan functionality. A capture-only DR is typically connected to a shadow register and data is read in parallel from the shadow register to the scan register within the CAPTURE-DR state. An update-only DR allows its data to be updated from the scan register to the shadow register within the CAPTURE-DR state. A capture/update DR allows data transferring in both directions. A scan-accessible DR is not connected to any shadow register, so the operations of capture and update are not needed.

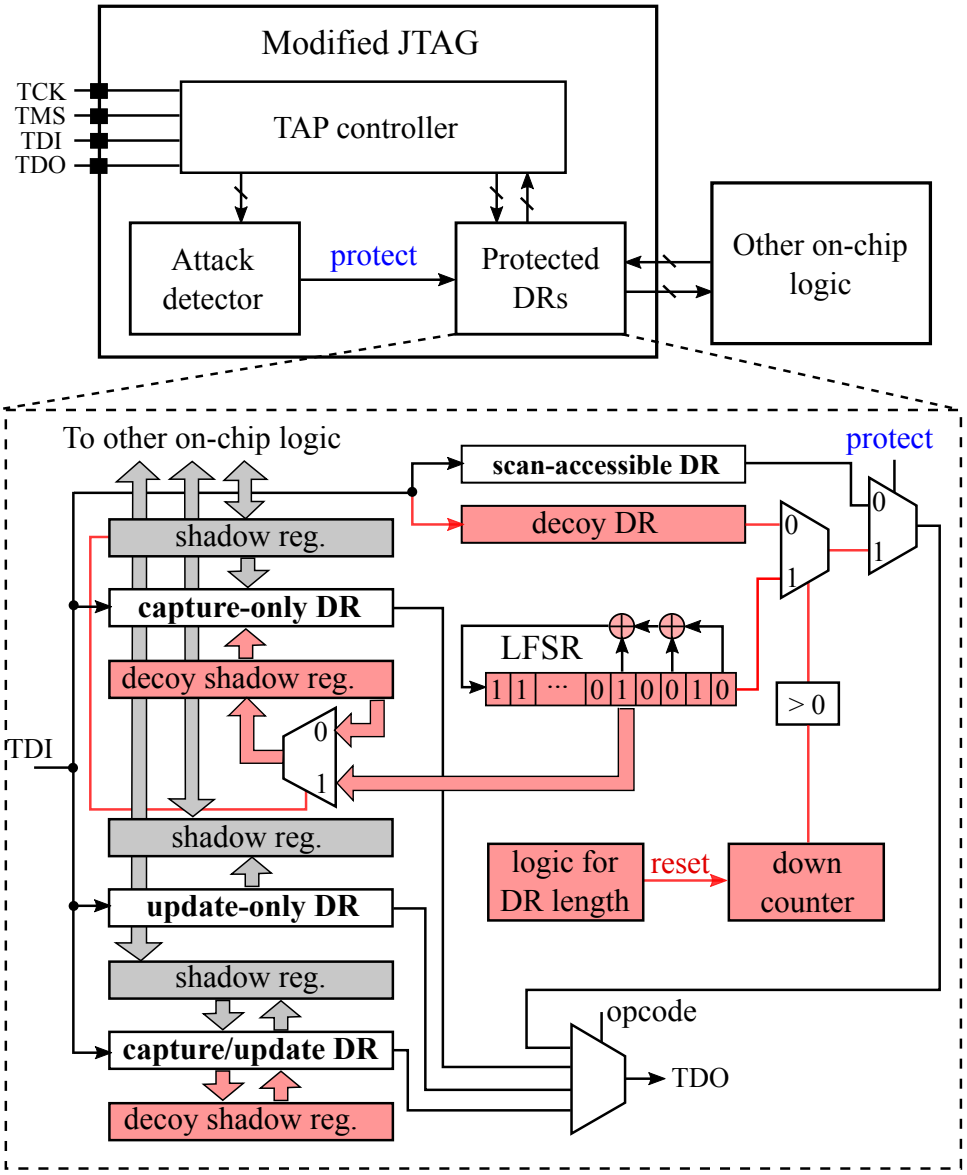


Figure 5.1: To protect the JTAG, access to DRs is modified as shown in red.

Table 5.1 lists how controllability and observability are prohibited for each type of DR. For a capture-only DR, a decoy shadow register and a linear-feedback shift register (LFSR) are employed to disable observability. More precisely, upon detection of an attack, connection to the actual shadow register is disabled; instead, the DR captures data from the decoy shadow register. Further, the data in this decoy shadow register is provided by the LFSR. An LFSR is commonly used as a pseudo-random number generator for stream ciphers due to its low hardware overhead, long

period, and uniformly distributed output stream [125]. The period of an LFSR is maximized if and only if its corresponding feedback polynomial is primitive. In this work, a 32-bit LFSR with a primitive-polynomial for feedback is implemented; its period is over 10^9. As shown in Figure 5.1, the hardware of an LFSR is a shift register whose leftmost bit is driven by the XOR of some bits of the shift register. The data in the decoy shadow register is updated by the LFSR only when the data in the actual shadow register is updated. The attacker may find that the data supplied via the TDO is random, but it is difficult to know when the data became random because the attacker does not know what to expect. Hence, the LFSR is effective in preventing the attacker from knowing which operation triggers the restriction. For an update-only DR, controllability is disabled through disabling the update operation. For a capture/update DR, a decoy shadow register is employed. Upon detection of an attack, connection to the actual shadow register is disabled; instead, the decoy register responds to the operations of capture and update. Because the decoy shadow register is stand-alone, it cannot control or observe any on-chip data. For a scan-accessible DR, because data should neither be shifted in nor out, the shift operation is disabled. A decoy DR and the LFSR are used instead to support the shift operation and supply data to the TDO. The decoy DR is a shift register that imitates the shift operation of the normal DRs. Once the JTAG enters the protection mode, the decoy DR and the LFSR, instead of the normal DR, are used for shifting data in and out. The selection of either the decoy DR or the LFSR relies on the comparison between the number of shifting clock cycles and the length (L) of the DR. The LFSR supplies bits to the TDO for the first L clock cycles, while the decoy DR provides bits after L clock cycles, allowing the serial input supplied by the user (via the TDI) to be observed at the TDO after an expected delay (i.e., L clock cycles). To achieve this function, the decoy DR needs to be as long as the longest scan-accessible DR and its length should always correspond to the DR that is being selected in real-time.

Obfuscation provided by the restricted JTAG access might be circumvented in two scenarios. In the first scenario, a read-after-write operation may exhibit inconsistent results, which can be noticed by the attacker. The second scenario involves the random bit stream supplied by the LFSR. However, these two scenarios do not necessarily reveal evidence of the JTAG under protection

because an attacker is assumed to have no prior knowledge of the private JTAG functions. In other words, the attacker neither recognizes an accidentally-executed read-after-write operation, nor knows that a selected DR should provide certain data rather than the observed bits.

5.2 Security Analysis

This section analyzes security of both the access restriction (described in Section 5.1) and the ML-based detection (described in Chapter 3). Particularly, two types of attacks are analyzed, namely, adversarial attack and disguised attack.

5.2.1 Adversarial Attack

The detector, including a feature collector, a ML classifier and an evidence collector, needs to be implemented on chip for detecting JTAG attacks. Different from password-based encryption, the detection is stand-alone, meaning that no extra steps (e.g., authentication) are needed before using the JTAG. Since the parameters of the detector need not be distributed to users, it is reasonable to assume that the attacker described by the model in Table 1.2 does not know the parameters of the classifier.

Despite of lack of knowledge, an attacker can attempt to learn sufficient information about the classifier in order to construct attacks that can escape detection. The work in [126] defines an *adversarial classifier reverse engineering* (ACRE) learning problem, where the goal of the attacker is not to perfectly find the classification boundary, but rather to identify high-quality instances that are not classified as attacks. The quality of an instance can be interpreted as the effectiveness of accomplishing the attack. Typically, when an attack is modified in order to escape detection, its effectiveness is compromised. This effectiveness is measured by a pre-defined adversarial cost function. For example, in the domain of email spam detection, spammers may learn to fool the classifier by inserting "non-spam" words into emails. The resulting email may not be effective in

TABLE 5.2: ATTACK MODEL AIMED AT THE DETECTOR (AS A SUPPLEMENT TO TABLE 1.2).

The adversary knows ...
✓ size of the sliding window w
✓ features used by the detector
✓ underlying algorithm (e.g., decision tree)
✗ parameters of the classifier and the evidence collector

selling their products as the original emails, but it can escape detection of the classifier. The work in [126] demonstrates that an attacker can find near-optimal evasion instances for a linear classifier, using a reasonable number of queries to the classifier. Further, the work in [127] generalized their result from linear classifiers to the family of convex-inducing classifiers that partition the space of instances into two parts, one of which is convex. Both methods, however, rely on a prerequisite that the attacker can query the membership for arbitrary instances. However, the detector developed in this work does not provide explicit membership for negative instances.

In addition to the attack model in Table 1.2, we make more assumptions for the attacker in terms of awareness of the detector, as exhibited in Table 5.2. The aim of the attacker is to uncover private JTAG functions and access the JTAG for other malicious purposes, without being detected. To construct disguised attacks, the attacker needs to learn sufficient information about the classifier. Based on the aforementioned analysis, this is typically achieved through querying the membership of specific instances.

To examine the possibility of querying the membership for arbitrary instances, two cases are considered. The first case assumes that the restricted JTAG access described in Section 5.1 is not employed. The test output instead provides zeros or ones upon detection of an attack. In this case, the attacker can observe when access to the JTAG is disabled, and further conclude that the most recent sliding window is classified as positive (see proof in Appendix A). Nevertheless, the membership of even earlier sliding windows are not revealed because labeling a user is decided by the evidence contributed by a series of sliding windows instead of individual ones. For example, a la-

beling of an attack might be caused by a few strong positive predictions, or a large number of weak positive predictions. The second case assumes that the restricted JTAG access is employed. As analyzed in Section 5.1, the attacker cannot identify the triggering sliding window, and consequently cannot obtain any more positive instances.

Hence, since the use of the HMM-based evidence collector and the restricted JTAG access prevents leakage of membership for both positive and negative instances, adversarial attacks to the detector become quite difficult. To reverse engineer the ML classifier, the attacker needs to make guesses for the membership of an instance.

5.2.2 Disguised Attack

In this subsection, it is further assumed that the attacker has already obtained some negative instances (i.e., legitimate sequences). Under this assumption, we will evaluate how likely it is the attacker can use these negative instances to disguise other attacks. The problem is modeled as follows. An attacker knows how to operate a portion of the private JTAG functions but not all. To discover the unknown functions, the attacker will attempt to disguise attacks through interleaving them with already-known legitimate sequences. For example, if the attacker loads one unknown instruction into the JTAG after every nine legitimate ones, then it may escape the detection because 90% of the instructions seem "legitimate". To verify this assertion, an experiment is performed using the OpenSPARC T2, as described next. The attacker is assumed to know how to operate control registers (shown as "creg" in Figure 3.15). To disguise the attack, a legitimate operation of the control registers (with sequence length $= 6$) is inserted into the attack traces after every segment of l instructions, where l varies from one to eight. The error rate of identifying disguised attacks is compared with the error rate of identifying legitimate operation and undisguised attacks, as shown in Figure 5.2. According to the results, attacks disguised by legitimate operation demonstrate higher probability of escaping detection. Although it is expected that back-and-forth jumps between unknown functions and known ones are likely to produce positive predictions, the

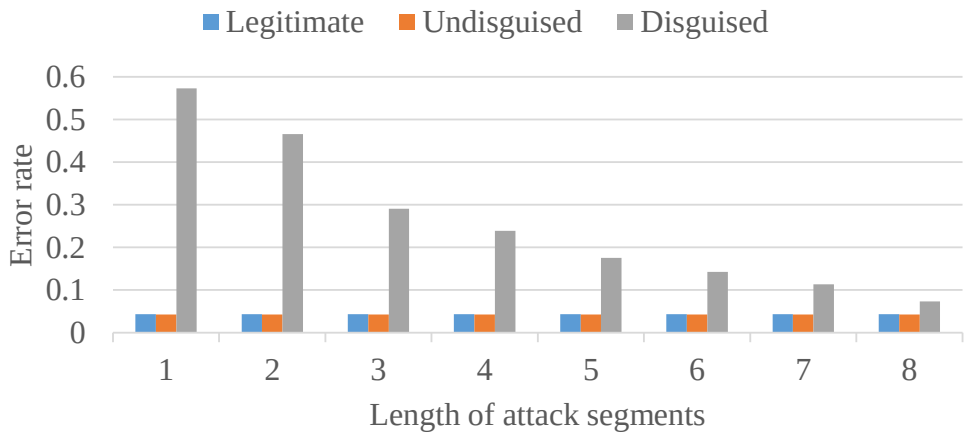


Figure 5.2: The error rate of identifying disguised attacks is evaluated using a decision-tree detector and compared with the error rate of identifying legitimate operation and undisguised attacks. An attack trace is disguised through segmenting the trace and then inserting legitimate sequences between the segments.

detector is still weakened by disguised attacks. It is worth noting that the high error rate resulted from short attack segments, as exhibited in Figure 5.2, is not always achievable. The first reason is that the length of attack traces varies. Some attack traces comprise only a few instructions (e.g., read data from a sensitive register), while others can be much longer. Second, inserting legitimate segments into an attack trace may compromise/damage the effectiveness of the attack. Hence, the detector, although likely to be weakened by disguised attacks, is still effective, especially for the attack traces that are long and/or unsplittable.

Even worse, if the attacker has full knowledge of the private JTAG functions, then he/she can operate the JTAG in a legitimate way, which cannot be detected. In another scenario, if the attacker has obtained the parameters of the detector (e.g., the tree nodes of the forest, the SVs and their α for the SVM, or the representative instruction sequences), then the attacker can also escape the detection by mimicking legitimate operation. Nevertheless, this is outside the attack model for this work.

5.2.3 Other Security Concerns

Once restricted, the JTAG will supply fake data, which however may cause a problem for legitimate users. An example involves using the JTAG for software debugging. If the JTAG is restricted due to a misclassification, then the user is provided with fake data without any notification. This will cause a problem if debugging data are supplied to other systems and/or used for further analyses. This problem can be mitigated by reducing occurrence of false positives. However, to further avoid risks caused by fake data (this might be significant in some scenarios), the JTAG should never be restricted. A possible solution is to authenticate the identity of the user.

Another security issue attackers may exploit is the proposed disabling of JTAG functionalities. In particular, an attacker could successfully execute a denial-of-service (DoS) attack by inserting malicious code into legitimate JTAG programs. This attack needs to be considered for achieving comprehensive security; however, detecting DoS attacks may need other features and/or models, which is beyond the scope of this dissertation.

5.3 Detector Implementation

This section elaborates upon hardware implementation of a sliding-window-based detector and feasibility of software implementation. As shown in Figure 5.3, the detector includes a feature collector, an ML classifier (may require non-volatile memory), a sanity-check function, and an evidence collector. These modules will be implemented and evaluated using the OpenSPARC T2 design.

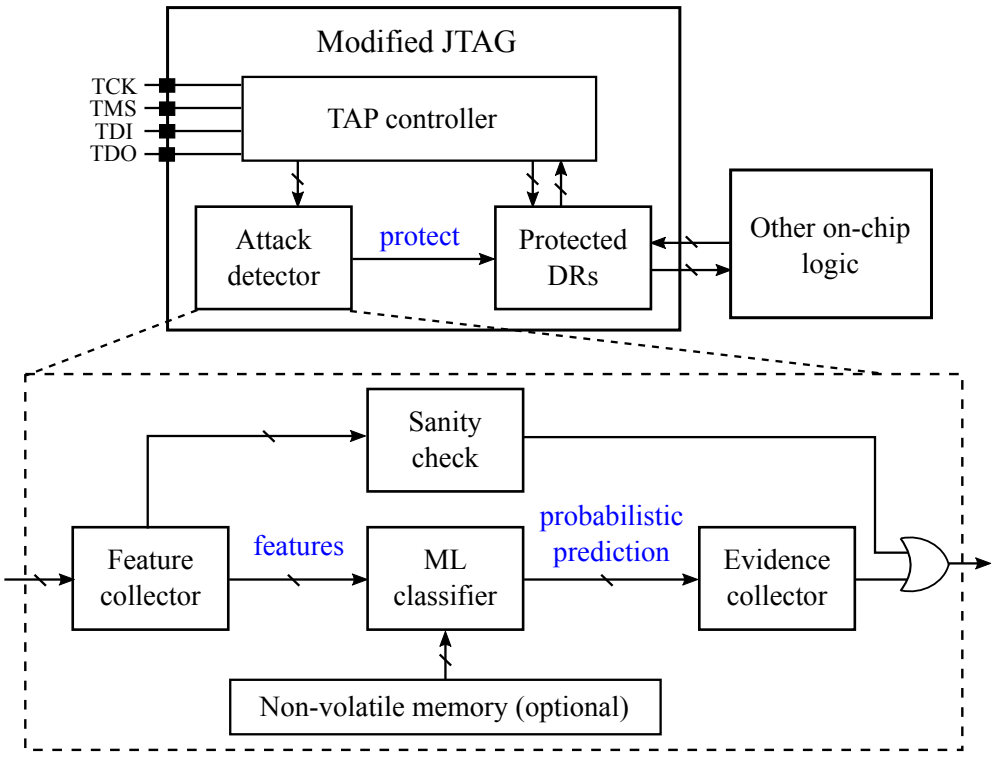
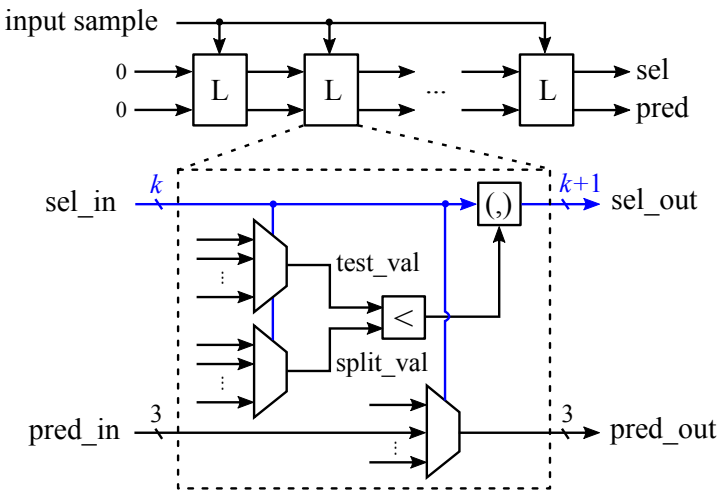


Figure 5.3: The overall architecture of the sliding-window-based detector.

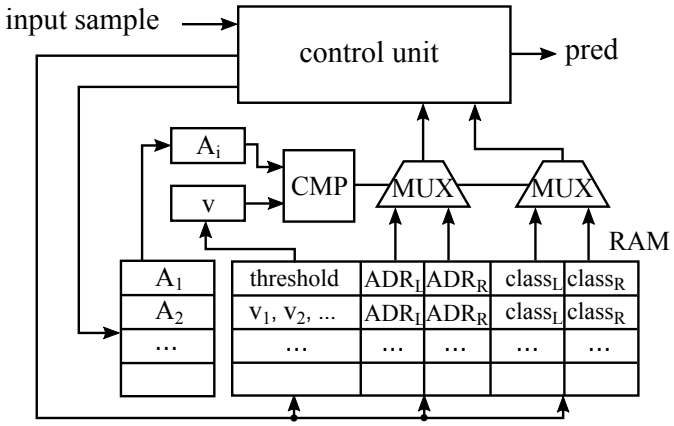
5.3.1 ML Classifiers

According to the results of cross-validation in Figure 3.11, several ML models, including a decision tree, a random forest, and an SVM, show similar performance in distinguishing legitimate operation and attacks. The implementation of these classifiers is discussed as follows.

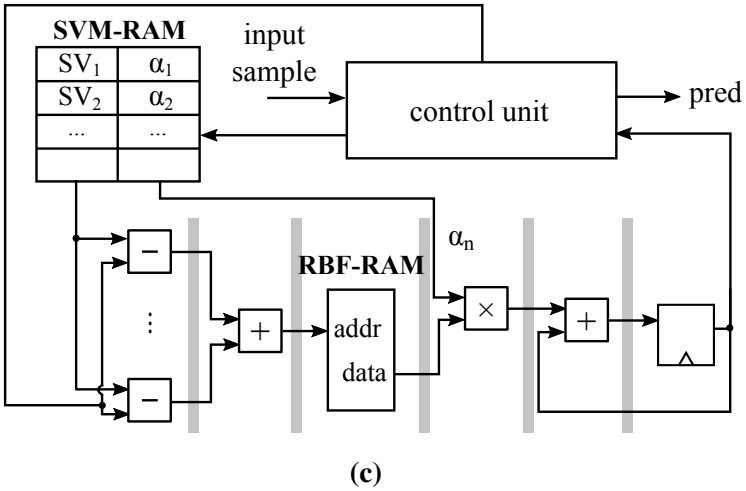
A decision tree can be implemented using combinational logic since it can be described as a hierarchy of "if-then" rules. Figure 5.4(a) shows the architecture of a combinational decision tree. Each block "L" in Figure 5.4(a) represents a level of tree nodes. The signal "sel_in" selects the specific tree node to be processed, including the feature to be evaluated (i.e., test_val) and the threshold stored in the tree node (i.e., split_val). Then, test_val and split_val are compared, with the result (together with the comparison results of previous levels) used for determining which tree node of the next level will be processed. The classification terminates at a leaf node where a three-bit result indicating the probabilistic prediction is provided. Note that the probabilistic



(a)



(b)



(c)

Figure 5.4: Architecture of (a) a combinational tree, (b) a sequential tree, and (c) an SVM.

prediction is decided by the fraction of samples of the same class within the leaf node. In this work, the probabilistic prediction ranges from 0 to 7, where a greater value means higher probability of attack. A combinational tree, although incurring small overhead, cannot be modified once implemented. If the designer needs to modify the classifier (this might be needed considering that the classifier is retrained based on newly-collected attacks), then the parameters of the decision tree needs to be stored in a memory, which results in a sequential implementation. As shown in Figure 5.4(b), the parameters of a tree are loaded into a RAM when the chip is powered on. Since the access to nonvolatile memory only occurs at the power-on stage, the online classification latency is not impacted. A tree unit employs the architecture of a universal tree node described in [128]. During the classification, one node is processed per clock cycle, so the required number of clock cycles is equal to the depth of the tree. It is noted that a continuous feature is compared with a threshold, while a discrete feature is compared with a set of values (i.e., v_1, v_2,...). In this work, the size of the RAM storing the learned tree is 2KB assuming at most 512 tree nodes.

A random forest is implemented as an ensemble of trees, using either combinational or sequential logic. In this work, a forest includes three trees, and its probabilistic prediction involves the average of the predictions provided by the trees. For the sequential implementation, the size of the RAM storing the learned forest relies on the number of trees and the size of each tree. In this work, a RAM with 3KB is used, assuming three trees and at most 256 nodes per tree (the trees within the ensemble are smaller than the tree that acts as a classifier individually).

Figure 5.4(c) shows the architecture of an SVM classifier. When the chip is powered on, the parameters of the learned SVM (i.e., each SV_i and each corresponding weight α_i) are loaded into a RAM (named SVM-RAM). The SVM classifier employs a fully pipelined architecture that consists of distance computation, an RBF kernel computation, a multiplication, and an addition. The distance computation employs L1-norm and the computation of the RBF kernel employs an LUT (loaded in RBF-RAM) that is built in advance by calculating all possible values. Note that the chip area required by an SVM classifier depends on w (i.e., size of the sliding window) in two ways. First, w affects the width of the pipeline linearly since it represents the dimension of data.

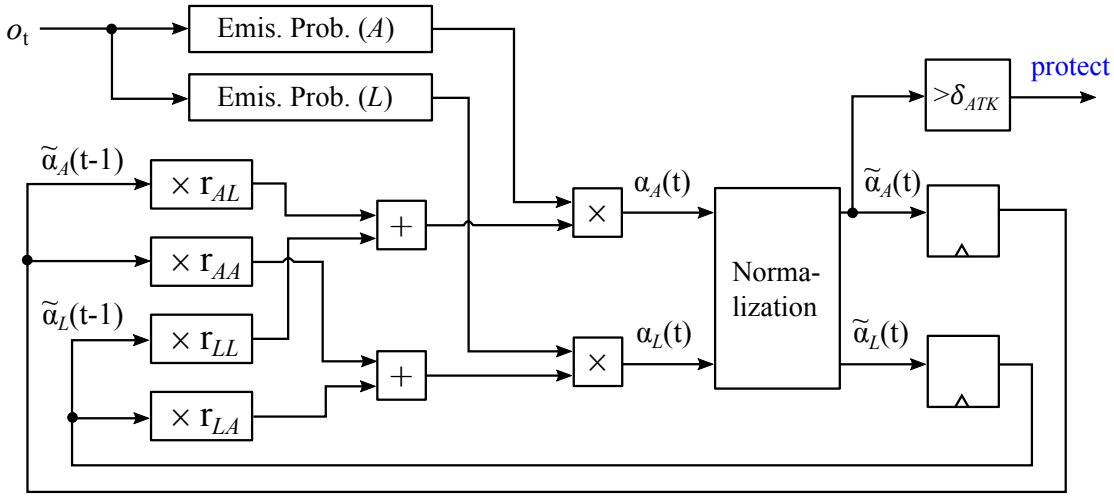


Figure 5.5: Architecture of the evidence collector.

Second, the size of the SVM-RAM relies on w because w affects both the number of SVs and the size of each SV. A larger w produces more SVs, that is, for example, the numbers of SVs produced by w = 3, 4, 5, 6 are 763, 844, 901 and 962, respectively. However, because these numbers are between 512 and 1024, the number of rows remains 1024 (each row stores an SV). Consequently, the area for the SVM-RAM is linearly dependent on w. The size of the RBF-RAM also depends on w linearly. To summarize, the relative area overhead in terms of w is: 0.75 ($w = 3$), 1 ($w = 4$), 1.25 ($w = 5$), 1.5 ($w = 6$).

5.3.2 Evidence Collector

As described in Section 3.3, the evidence collector employs an HMM with two hidden states (i.e., S_L and S_A). The evidence collector aims to calculate the probability of S_A using a lattice-like process shown in Figure 3.4. As shown in Figure 5.5, the architecture of the evidence collector contains multiplications and normalizations. Nevertheless, since all the operators, except for the three-bit prediction provided by the classifier, have eight bits, the overhead caused by multiplication and normalization is tolerable.

TABLE 5.3: DETECTORS BASED ON DIFFERENT CLASSIFIERS ARE COMPARED (GATE EQUIVALENT = TOTAL AREA / AREA OF A TWO-INPUT NAND GATE).

	Hardware					**Software**		
	Tree (comb.)	Forest (comb.)	Tree (seq.)	Forest (seq.)	SVM	Tree	SVM	RNN
RAM	-	-	2KB	3KB	2.94KB	-	-	-
Area (gate equivalent)	4,286	6,286	176,170	275,266	223,298	-	-	-
% of OpenSPARC T2 area	0.021%	0.031%	0.869%	1.36%	1.1%	-	-	-
Area compared to JTAG	0.66×	0.97×	27.2×	42.4×	34.41×	-	-	-
Latency (clock cycles)	5	5	23	17	520	750	5,780	28,900
Dynamic power (mW)	0.093	0.144	7.62	11.6	9.45	-	-	-
Error rate for detection	5.8%	7.3%	5.8%	7.3%	14.1%	5.8%	13.6%	2.2%
Updatable	No		Yes			Yes		

5.3.3 Overhead Compared to Software

The detectors are synthesized using the commercial logic synthesis tool Design Compiler with a 0.18 µm library (by Synopsys) [129]; synthesis results are given in Table 5.3. The results show that the sequential implementation of the decision tree is 40 times larger than a combinational tree, excluding the overhead of non-volatile memory. The random forest even demonstrates a 43 times difference. The latency of the combinational tree and forest is five clock cycles, including one for feature collection, two for classification, and two for evidence collection. Note that both the classification and the evidence collection take two clock cycles, because registers are inserted into the tree and the evidence collector in order to shorten their critical path. For the sequential tree and forest, since each tree node is processed per clock cycle, the classification requires many more clock cycles. The SVM classifier requires the most clock cycles due to a serial, pipelining processing of support vectors.

It is noted that the SVM classification results computed by hardware may differ from software due to limited precision for α_i and the RBF values. Figure 5.6 shows that a higher precision reduces the mean squared error (MSE) for the classification results (compared to software), but at the cost

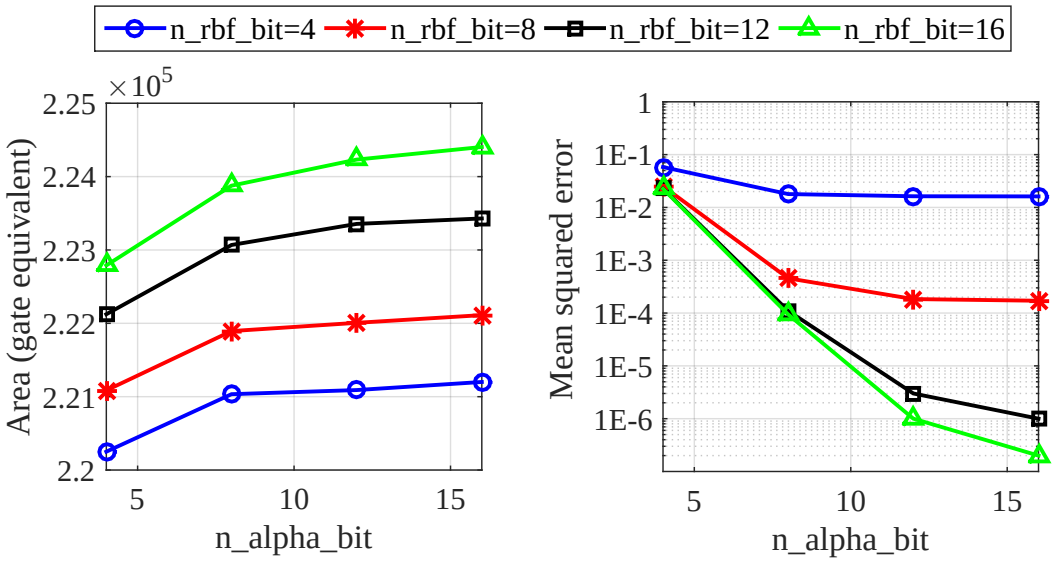


Figure 5.6: Area of the SVM detector and the mean squared error of the classification results (compared to software) are impacted by the number of bits for α_i (n_alpha_bit) and the RBF values (n_rbf_bit).

of more chip area. In the implementation, both α_i and the RBF value employ 12-bit, fixed-point representations, which can reduce the MSE significantly to a low level (3×10^{-6}) when compared to the representations with fewer bits.

All of the detectors can also be implemented using software, i.e., classification and evidence collection are computed using a software program executed within a processor. The JTAG of the LEON3 design, as shown in Figure 2.4, serves as an example. Recall that the LEON3 processor implements a debug mode during (described in Section 2.1.2) which the pipeline is idle and the processors are controlled through a debug interface (e.g., the JTAG). Therefore, executing the detection algorithm using software is possible if one or several processors are properly configured for this purpose. Since no additional processor is needed for executing the detection algorithm, the overhead of chip area incurred by software implementation is not considered in Table 5.3. To estimate the execution time, a decision tree and an SVM are implemented using the C++ interface of the OpenCV [130]. According to experiments using an Intel Core i7 with a 3.4 GHz clock, classification takes 2.2×10^{-7}s for the decision tree and 1.7×10^{-6}s for the SVM. This latency

is interpreted as number of clock cycles as exhibited in Table 5.3, which is much longer than hardware.

The detection algorithm may also be executed within a remote processor. In this case, the data collected from JTAG operation need to be transferred to a remote processor, and the resulting detection result needs to be sent back to the chip. To estimate the communication time, we connect a local computer to a variety of remote servers located within both LAN and WAN, using the `ping` command[1]. The resulting communication time ranges from 1 ms to 200 ms, which however is too large compared the execution time. Therefore, a remote processor is not suggested to be used for online detection.

Based on the results listed in Table 5.3, the designer is suggested to choose the ML algorithm and its corresponding implementation based on complexity of the JTAG functions and the trade-off among area, latency, performance of detection, and the need to upgrade. The area of the detectors might be acceptable compared to the whole design, considering that the OpenSPARC T2, due to its complex testing/debugging functions, is likely to require a complex detector. Although the sequential detectors are much larger than the JTAG, it may be worthwhile because the detector aims to protect the whole system rather than only the JTAG (the JTAG serves as an interface for the system). Moreover, this area difference is expected because a classical JTAG, consisting of a TAP controller and DRs (excluding internal scan chains), is typically quite small. In addition to area, latency also needs to be considered carefully because attacks to JTAG/IJTAG can happen very quickly (e.g., a few dozen clock cycles). To detect these attacks, the detector should be as fast as possible. According to the latency numbers shown in Table 5.3, a hardware implementation is significantly better than software. Table 5.3 indicates that the software-based detector consume hundreds or thousands more clock cycles than hardware; nevertheless, a software-based detector may still be used because they can typically run with a much faster clock than the clock for JTAG. In terms of detection accuracy, a decision tree, a random forest, and an SVM demonstrate similar

[1]The `ping` command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to a destination computer and waiting for a response [131].

performance (although the experiments for the OpenSPARC T2 shows that a decision tree outperforms the others). An RNN achieves the best performance but at the cost of longer latency. To obtain the best detection accuracy, an RNN should be used. Another consideration involves the upgrade of detectors to deal with future types of attacks. This requires that the parameters of the detector are stored in non-volatile memory (e.g., firmware), also implies that a combinational decision tree or random forest cannot be used. The feasibility of upgrading a detector will be discussed further in Section 5.4.

All of the aforementioned issues, including the complexity of the JTAG functions, area, latency, performance of detection, and the need to upgrade, need to be considered for choosing the most suitable detector. The first scenario involves multi-core processors (such as the OpenSPARC T1, T2 and the LEON3 processors) that are typically equipped with a variety of JTAG functions for testing and debugging. If an on-chip core is available and detection accuracy is the primary goal, then software implementation of an RNN can be chosen. Another scenario involves IoT devices. Different from multi-core processors, an IoT device is typically used for a specific application, thus resulting in simple, repeated operation [132]. To detect an attack of an IoT device, a combinational decision tree may suffice. Other than monitoring the JTAG ports of individual devices, the traffic within the network should also be monitored and analyzed, an interesting topic for future research.

5.4 Discussion

This section discusses two issues concerning detector implementation. The first issue involves scalability of the detectors. That is, as new types of attacks emerge, the performance and the size of the detectors may be impacted. We conduct the following experiment as a preliminary evaluation. First, attacks S_4 to S_9 listed in Table 2.2 are gradually added to the training set for both the random forest and the SVM. Next, the performance of both detectors are evaluated using ten-fold cross-validation. The size of the random forest is evaluated using the average number of nodes per tree, while the SVM size is evaluated using the number of support vectors. The results in Figure 5.7

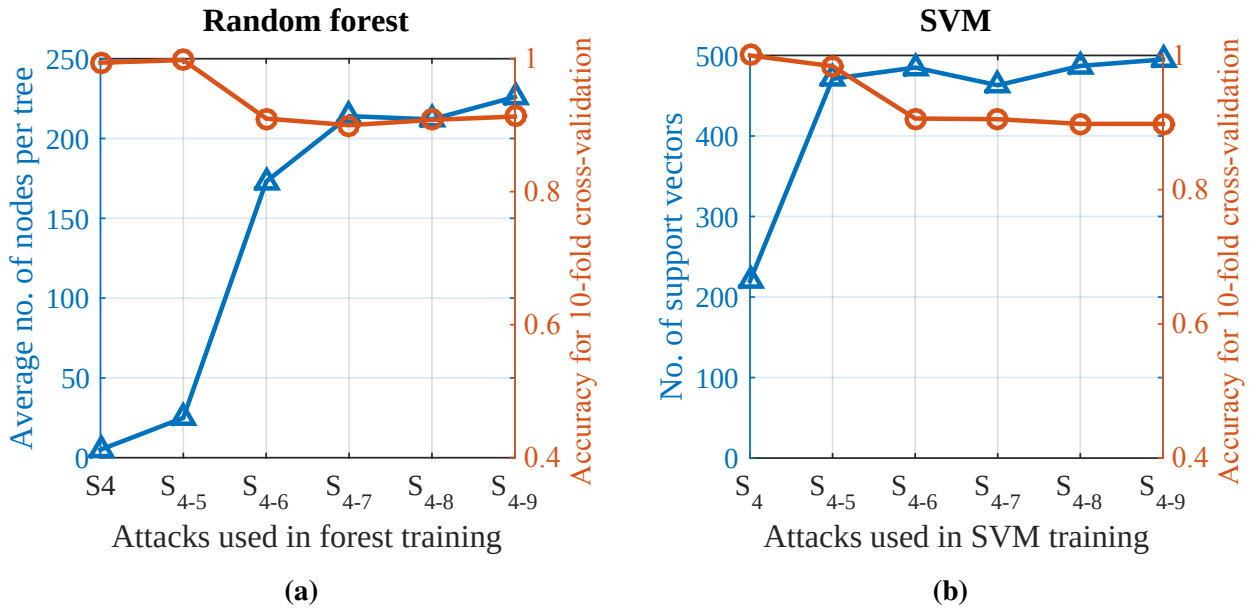


Figure 5.7: Performance and size of (a) the random-forest-based detector and (b) the SVM-based detector are impacted by the variety of attacks used in classifier training.

demonstrate an overall increase of size and a decrease of performance as more types of attacks are used in classifier training. However, the impact of different attacks varies. The size and the accuracy change dramatically for the first two types of attacks (i.e., S_4 and S_5); then they become more stable as the variety of attacks increases.

The second issue involves the update of detectors in case that new attacks are collected. This requires that the parameters of the detector are stored in non-volatile memory (e.g., firmware), such that a combinational decision tree or random forest cannot be used. When new attacks are collected, it needs to be checked if they can be effectively detected by the existing detector with high confidence. If not, the detector might be retrained and updated via the on-chip memory. This can be achieved by incorporating the parameters of the detector into firmware that is typically upgraded via USB [133] or JTAG [134]. To ensure security of firmware upgrade, two issues need to be considered. First, only authorized firmware can be upgraded. This can be realized using firmware signature [135], i.e., generating a unique hash code for each authorized copy and appending it to the firmware when distributed. The other issue involves leakage of firmware contents. This can be

mitigated by encrypting the firmware using cryptographic blocks [45]. The decryption of firmware needs to be performed on chip, which however requires additional chip area. Since firmware, as well as its encryption/decryption, is ubiquitous for electronic systems, the overhead caused by incorporating detector parameters into firmware can be amortized.

5.5 Summary

This chapter describes a secure JTAG architecture, which not only restricts access to the JTAG through prohibiting observability and controllability provided by the JTAG, but also hinders adversarial attack to the ML model through obfuscating the JTAG operation triggering the detection. This architecture is implemented within the OpenSPARC T2 and demonstrates moderate overhead (i.e., 22% chip area compared to the original JTAG and 0.01% of the T2). This chapter also describes hardware implementation of the attack detectors that employ various ML algorithms, and evaluates their overhead using the commercial logic synthesis tool Design Compiler by Synopsys. According to the synthesis results, the combinational implementation of a decision tree and a random forest leads to small overhead (even smaller chip area than the original JTAG). The sequential implementation of a decision tree, a random forest, and an SVM, although incurring much larger overhead, allows the detector to be updated (e.g., when new attacks are collected and the classifier is retrained). In addition, the detector may also be implemented using software if the system has a processor core. To summarize, a designer is suggested to choose the ML algorithm and its corresponding implementation based on the trade-off between overhead, performance, and the need to upgrade.

Chapter 6

Summary and Future Work

JTAG and its related standards are primarily used for facilitating IC testing and design debug. However, because JTAG needs to be left operational after fabrication, for the purpose of in-field debugging and programming, it inevitably provides a "backdoor" that can be exploited by illegitimate users. Previous approaches, such as fusing-off the JTAG port and encrypting JTAG access, have been proposed to prevent illegitimate access to the JTAG. However, fusing-off the JTAG also permanently disables in-field debugging, while encrypting JTAG access typically suffers from eavesdropping of the key or requires availability of a network. In this work, we propose to ensure security of the JTAG through real-time detection of illegitimate access to the JTAG using machine learning (ML). We also develop a secure JTAG architecture that can restrict access to the JTAG upon detection of an attack. Compared to previous work, the ML-based attack detection not only preserves the functions of in-field debugging/programming, but also minimizes the risk of key eavesdropping. The methodology and experiments presented in this work demonstrate the effectiveness of the ML-based attack detection and the access restriction, and further verify the feasibility of implementing the methodology on chip for online detection and restriction. In addition, this dissertation analyzes security that the developed method provides, and characterizes its vulnerability.

JTAG protection approaches, including previous work and this work, are developed assuming specific attack models. Hence, the designer is advised to assess the security specification for the IC, and then adopt the approaches that best satisfies the outcomes of the assessment. Moreover, the ML-based attack detection, orthogonal with previously-reported approaches, can be combined with them (such as encryption) for achieving complementing JTAG protection.

6.1 Dissertation Contribution

This dissertation has presented a JTAG protection method, including ML-based detection of JTAG/IJTAG attacks and a restriction of JTAG access when attacks are detected. The main contributions of both aspects are summarized next.

Characterization of JTAG Operation

- **JTAG operation and attacks** - This work considers a standard JTAG architecture and a bus-based debug architecture. Then, three representative designs, including the OpenSPARC T1, T2 and the LEON3 processors, are studied, with a set of legitimate JTAG operations created. In addition, existing non-intrusive attacks to the JTAG are summarized, based on which attack traces are created, mimicking reverse engineering of the private JTAG functions.

- **Features** - JTAG operation is characterized using four categories of features. The features monitor the instruction register, the data registers, and the TAP controller, while the last category of features aim to improve classification accuracy through incorporating additional information. Then, effectiveness of these features is evaluated, with relevant features selected for attack detection.

JTAG Attack Detection

- **Sliding-window-based detection** - JTAG operation is monitored and collected using a fixed-size window that behaves in an overlapping manner (called a sliding window). The use of

a sliding window not only allows a variety of ML algorithms to be used for the following classification, but also benefits online detection because it allows a ML model with less complexity, such as a decision tree, to be implemented on chip. In addition, a regularized Kullback-Leibler divergence is used for measuring similarity between legitimate JTAG operation and attacks, based on which the size of the sliding window is determined. Experiments using a decision tree demonstrate an error rate of 3% and 5%, for the OpenSPARC T1 and T2, respectively.

- **Delayed labeling** - Due to the variance that naturally occurs in JTAG operation, classification based on a single sliding window might not be reliable. To mitigate this problem, a hidden Markov model is employed for collecting evidence of an attack, such that the user is not labeled until sufficient evidence has been collected.

- **Sequence models** - Sequence-based models, including recurrent neural networks and hidden Markov models, are evaluated. Compared with sliding-window-based detectors, sequence models reduce the error rate to even lower levels (i.e., 1% and 3%), possibly because they can better capture serial dependence within JTAG operation.

- **Detection of unseen attacks** - A concept of open-space risk is used for modeling the inaccuracy of detection caused by new attacks that typically appear as a cluster far from known attacks. To minimize the open-space risk, a cascade model, combining a one-class SVM and a binary SVM, is developed, both of which demonstrate an improvement in attack detection for an open space.

Detection of IJTAG Attacks

- **Features** - Different from a standard JTAG, access to an IJTAG network involves configuring a hierarchy of SIBs/SCBs and setting/resetting a large number of TDR bits. Correspondingly, IJTAG operation is characterized using features that describe the value stored in the SIBs/SCBs and the TDRs.

- **Feature reduction** - The features characterizing IJTAG operation are likely high-dimensional, which poses a challenge for on-chip detection. A low-density parity-check (LDPC) based compression method is developed for reducing the dimension of data. According to experiments based on a modified version of the OpenSPARC T2, the use of LDPC-based feature reduction eliminates 91% of the features and reduces circuit size by 43% without affecting detection accuracy.

Restriction of JTAG Access and Implementation of Detectors

- **Restriction of JTAG access** - An architecture of secure JTAG is developed, which can restrict access to the JTAG upon detection of an attack. The secure JTAG not only prohibits observability and controllability provided by the JTAG, but also obfuscates the operation triggering the restriction, which consequently hinders adversarial attack. The architecture, implemented within the OpenSPARC T2 design, incurs moderate overhead (i.e., 22% chip area compared to the original JTAG and 0.01% of the T2).

- **Security analysis** - The use of the secure JTAG architecture and the HMM-based evidence collection prevents an attacker from querying the membership for arbitrary instances, therefore making the ML-based detector robust to adversarial attacks.

- **Hardware implementation of detectors** - The feasibility of a sliding-window-based detector is verified through implementing it within the OpenSPARC T2 design and evaluating the resulting overhead. Various ML algorithms are implemented using either combinational or sequential logic, which results in different circuit size and latency. The designer is advised to choose the ML algorithm and its implementation (either combinational or sequential) according to the trade-off between detection accuracy, overhead and the need to upgrade.

6.2 Future Work

Listed below are some topics to be explored in future work, for improving the JTAG protection method developed and analyzed in this dissertation.

- **Generating JTAG attacks automatically** - Overhead of the ML-based attack detection not only includes circuit size and latency, but also includes the effort of collecting attacking data for constructing the ML model. Although the ML-based attack detection is a generic method for JTAG, the detection model trained for a given JTAG design cannot be used for another JTAG design directly. Thus, it is beneficial to automate the process of generating JTAG attacks. This is possible because the attack strategies listed in Table 2.2 are also generic for uncovering the private JTAG functions.

- **Hardware and software co-detection** - As shown in Section 3.6.3 and 3.6.4, recurrent neural networks (RNNs) demonstrate extremely high accuracy in detecting JTAG attacks, and even unknown attacks. Even so, an RNN is more suitable to run in software because implementing an RNN in hardware can incur intolerable overhead. A possible idea is to combine a hardware detector (e.g., a combinational decision tree) and a software RNN for achieving comprehensive detection, as shown in Figure 6.1. A combinational tree incurs small on-chip overhead, which is suitable for online detection, while a software RNN, though slow, can provide more accurate prediction.

- **Re-enabling of JTAG access** - In this work, access to the JTAG is restricted permanently upon detection of an attack, which however causes inconvenience for legitimate users in case of false positives. Although the use of HMM-based delayed labeling has reduced false positives effectively, it would be beneficial if a false positive can be properly addressed. An intuitive solution is to permit a legitimate user to re-enable access to the JTAG. The mechanism of re-enabling the JTAG should be sufficiently secure however. Recall our underlying assumption that the attacker does not know how to operate the JTAG legitimately. Thus, it

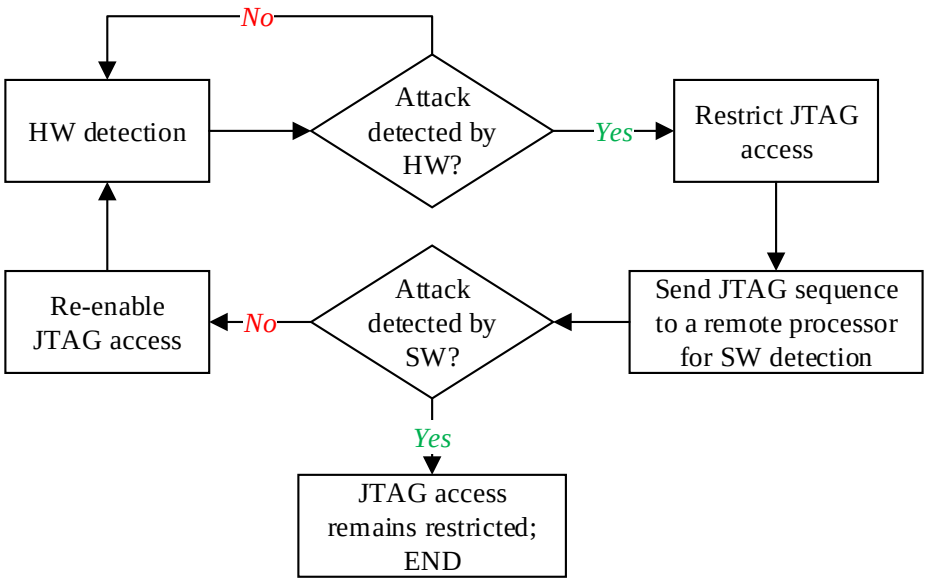


Figure 6.1: A combination of hardware and software for attack detection.

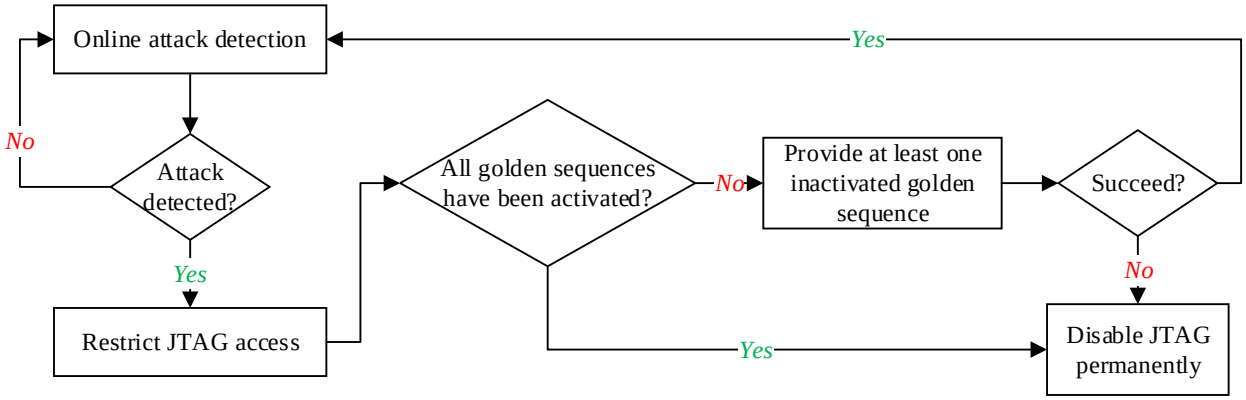


Figure 6.2: A mechanism that permits legitimate users to re-enable access to the JTAG.

might be possible to use legitimate JTAG sequences as a "key" to re-enable access to the JTAG, as illustrated in Figure 6.2. Specifically, a set of legitimate JTAG sequences (named golden sequences) are stored in chip. Every time an attack is detected and the JTAG is restricted, the user can re-enable it through operating a golden sequence that has never been operated before (once a golden sequence is operated, it is activated).

- **ML model retraining** - Although the detectors demonstrate a potential of detecting unknown attacks, it still does not guarantee that all types of attacks can be detected. We suggest

that hardware companies build a database of JTAG attacks and collect potential attacks using similar strategies as anti-virus software, such as analyzing user report and periodic security examination. When new attacks are collected, the detector is re-trained and updated to the on-chip memory. This can be achieved by incorporating the parameters of the detector into firmware that is typically upgraded via USB [133] or JTAG [134]. The security of firmware upgrade needs to be ensured, i.e., only authorized firmware can be upgraded [135] and the firmware needs to be encrypted [45].

- **Scalability of IJTAG protection** - An IJTAG-compatible system may have a large number of instruments. Even though the LDPC-based feature compression can significantly reduce the number of features, collecting data from each instrument can incur significant wiring overhead if the SIB/TDR values are tapped from each instrument to the global detector. This problem can be mitigated by 1) compressing the SIB/TDR values within the instrument before transmission to the detector, or 2) implementing an additional monitor that decodes the test input directly into the data to be loaded into the SIBs and TDRs, without accessing the SIBs and TDRs.

- **Improving detection accuracy for bus-based debug architecture** - The experiments using the LEON3 processor shows that a very large window ($\sim$40 operation cycles) is needed for distinguishing attacks from legitimate operation, resulting in data with high dimension. Even so, the ML-based detectors demonstrate poor accuracy, revealing that detecting anomalies in bus traffic is more challenging than monitoring the JTAG TAP. Thus, future work involves seeking more features and building more complex detection models for identifying attacks aimed a bus-based debug architecture.

Appendix A

Adversarial Analysis

Assume a sliding-window-based detector that involves a binary ML classifier and an evidence collector. The binary classifier gives a probabilistic prediction o_t for operation cycle t, while the evidence collector employs an HMM described in Section 3.3. For the state transition probabilities of the HMM, it is assumed that $v > 0.5$, meaning that the hidden state (either S_L or S_A) is more likely to remain unchanged, rather than to transfer to the other state. For the emission probability, it is assumed that the probability distributions for S_L and S_A are monotonically decreasing and increasing, respectively, and that they intersect at $o = 0.5$.

Theorem: If the detector labels the user as attack at operation cycle t, i.e.,

$$\widetilde{\alpha}_A(t) \geq \delta_{ATK} \quad \text{and} \quad \widetilde{\alpha}_A(t-1) < \delta_{ATK} \tag{A.1}$$

then the probabilistic prediction provided by the classifier at t must be illegitimate, i.e.,

$$o_t > 0.5 \tag{A.2}$$

Proof:

Since the evidence accumulator employs an HMM with two states (i.e., S_L an S_A), the probability of the hidden state being S_L and S_A can be computed using a forward procedure.

$$\alpha_L(t) = [v \cdot \alpha_L(t-1) + (1-v) \cdot \alpha_A(t-1)] \cdot b_{Lo_t} \tag{A.3}$$

$$\alpha_A(t) = [v \cdot \alpha_A(t-1) + (1-v) \cdot \alpha_L(t-1)] \cdot b_{Ao_t} \tag{A.4}$$

Let $\rho_t = \widetilde{\alpha}_L(t)/\widetilde{\alpha}_A(t)$, $\xi_t = b_{Lo_t}/b_{Ao_t}$, and $u = v/(1-v)$, and then we have

$$\begin{aligned} \rho_t = \frac{\widetilde{\alpha}_L(t)}{\widetilde{\alpha}_A(t)} = \frac{\alpha_L(t)}{\alpha_A(t)} &= \frac{v \cdot \alpha_L(t-1) + (1-v) \cdot \alpha_A(t-1)}{v \cdot \alpha_A(t-1) + (1-v) \cdot \alpha_L(t-1)} \cdot \frac{b_{Lo_t}}{b_{Ao_t}} \\ &= \frac{1 + u\rho_{t-1}}{u + \rho_{t-1}} \cdot \xi_t = \xi_t + \frac{(u-1)(\rho_{t-1}-1)}{u+\rho_{t-1}} \cdot \xi_t \end{aligned} \tag{A.5}$$

If the detector labels the user as an attacker at operation cycle t, then ρ_{t-1} and ρ_t must satisfy

$$\rho_t < \rho_{t-1} \quad \text{and} \quad \rho_t < 1 \tag{A.6}$$

Case 1: When $\rho_{t-1} \leq 1$, to satisfy

$$\begin{aligned} \rho_t - \rho_{t-1} = \frac{1 + u\rho_{t-1}}{u + \rho_{t-1}} \cdot \xi_t - \rho_{t-1} &= \frac{1 + u\rho_{t-1}}{u + \rho_{t-1}} \cdot \xi_t - \rho_{t-1} \cdot \xi_t - \rho_{t-1} \cdot (1 - \xi_t) \\ &= \frac{1 - \rho_{t-1}^2}{u + \rho_{t-1}} \cdot \xi_t - \rho_{t-1} \cdot (1 - \xi_t) < 0 \end{aligned} \tag{A.7}$$

We need to guarantee

$$\xi_t < 1 \tag{A.8}$$

Case 2: When $\rho_{t-1} > 1$, to satisfy

$$\rho_t = [1 + \frac{(u-1)(\rho_{t-1}-1)}{u+\rho_{t-1}}] \cdot \xi_t < 1 \tag{A.9}$$

We also need to guarantee

$$\xi_t < 1 \tag{A.10}$$

To summarize, if the detector labels the user as an attacker after operation cycle t, then

$$b_{Lo_t} < b_{Ao_t} \tag{A.11}$$

Because the emission probability distributions for S_L an S_A are monotonically decreasing and increasing, respectively, and they intersect at $o_t = 0.5$, we can conclude that $o_t > 0.5$, meaning that the probabilistic prediction provided by the classifier for operation cycle t must be illegitimate.

Glossary

A A $m \times n$ matrix that can compress an n-dimensional vector x to an m-dimensional vector z ($n >> m$). xix, 77–80, 84

B Emission probabilities of an HMM. 47, 48

C A coefficient that controls the trade-off between the slack variable penalty and the margin, used in SVM training. 42

F_{dr} Features that characterize the data loaded into the data register (DR) that is selected by a specific opcode. 26, 27

F_{ext} Features that require information outside an operation cycle. 26–28

F_{fsm} Features that characterize state traversal of the TAP finite state machine. 26, 27

F_{ir} Features that characterize the opcode loaded into the instruction register (IR). 26, 27

G A statistical model that maps an observation x to its most probable class label y. 40, 43

N_S Number of SIB/SCB bits in an IJTAG network. 76, 80

N_T Number of test data registers (TDRs) in an IJTAG network. 76, 80

N_{SV} Number of support vectors resulting from SVM training. 42

N_{obs} Number of observations collected from legitimate JTAG operation and attacks. 39, 42

$P(x|y = A)$ Probability distribution for an observation x, given that the user is an attacker. 38, 39

$P(x|y = L)$ Probability distribution for an observation x, given that the user is legitimate. 38, 39

$P_A(x)$ Probability that an observation x is an attack based on an SVM model. 55

$P_L(x)$ Probability that an observation x is legitimate based on an SVM model. 54, 55

$P_O(x)$ Probability that an observation x is inlier based on a one-class SVM model. 54, 55

R State transition probabilities of an HMM. 47, 48

S_A A hidden state within the HMM (used for collecting evidence of attack), representing that the actual user is an attacker. xviii, 46, 47, 49, 101, 115, 117

S_L A hidden state used by HMM (used for collecting evidence of attack), representing that the actual user is legitimate. xviii, 46, 47, 49, 101, 115, 117

S_{SV} The set of support vectors resulting from SVM training. 42

W A matrix of weights used for calculating the class label when SVMs or neural networks are used for classification. 42, 51

Γ_r A coefficient used by an RNN, which represents relevance of the previous neuron state c_{t-1} relative to the current observation x_t. 50, 51

Γ_u An update coefficient used by an RNN, which decides the current neuron state c_t based on the previous state c_{t-1} and a state candidate $\widetilde{c}_t$. 50, 51

α_i The i-th weight corresponding to the i-th support vector within a trained SVM. xx, 100, 102, 103

δ_{ATK} A threshold that determines if sufficient evidence has been collected for labeling the user. 46, 58, 115

δ A regularization term used for searching for the optimal window size. 39, 56

$\hat{\lambda}$ The parameters of an HMM that are estimated using maximum likelihood estimation. 51

$\hat{x}$ A vector reconstructed from a compressed vector z (x is compressed to z using a matrix A, i.e., $z = Ax$). 78

ι_L A binary variable that indicates if an observation x is anomaly. 55

λ_τ A threshold that determines the degree of anomaly for an observation x. 55

λ Parameters that define an HMM, including initial probabilities, state transition probabilities, and emission probabilities. 46–48, 51

π Initial probabilities of an HMM. 47, 48

σ A softmax function. 51

$\widetilde{P}(x|y = A)$ Regularized probability distribution for an observation x, given that the user is an attacker. 39

$\widetilde{P}(x|y = L)$ Regularized probability distribution for an observation x, given that the user is legitimate. 39

$\widetilde{\alpha}_A(t)$ Likelihood that the hidden state of an HMM at time t is S_A, i.e., an attacker. xvii, 46, 47, 115, 116

$\widetilde{\alpha}_L(t)$ Likelihood that the hidden state of an HMM at time t is S_L, i.e., legitimate. xvii, 46, 47, 116

ξ_i A slack variable (corresponding to the i-th observation x_i) that allows x_i to be misclassified with a penalty, used by SVMs with a soft margin. 42

b A bias term used for calculating the class label when SVMs or neural networks are used for classification. 42, 51

c_t The state of neurons at time t within a recurrent neural network (RNN). 49, 50

d Number of ones in each column of matrix A (used for compressed sensing). xix, 79, 80, 84, 87

g Local girth for a low-density parity-check (LDPC) matrix. xix, 78–80, 84

$k(x,x')$ A kernel function in terms of x and x' used in SVM training and classification. 42

k Number of nearest neighbors involved in k-NN. 43, 58

m Dimensionality of a vector z after compression. 78, 79, 84, 85, 87

n Dimensionality of a vector x before compression. 78, 87

o Observation defined for the HMM (used for collecting evidence of attack), representing the likelihood of an attack that is produced by the ML classifier, i.e., $P(y = A|x)$. 46, 115

q For a representative-based anomaly detector, JTAG operation is labeled as an attack only if the global score stays saturated for more than q consecutive operation cycles. 45, 58

v Probability that defines state transition within the HMM (used for collecting evidence of attack), either from S_L to S_A, or from S_A to S_L. xviii, 47, 49, 115, 116

w Size of a sliding window, i.e., number of operation cycles included in a sliding window. xviii, 37–39, 56, 57, 77, 80, 85, 100, 101

x_t The window of JTAG operation observed at time t. 50

x An observation collected from JTAG operation using a sliding window. 38–40, 42, 43, 54, 55, 77, 78

y_t Predicted label for a user at time t. 43, 50

y Predicted label for a user, i.e., either legitimate (L) or attack (A). 38–40

z A low-dimensional vector resulting from compressing a high-dimensional vector x. 78

Bibliography

[1] "D-Link DIR-620 Router." https://openwrt.org/doku.php?id=ru%3Atoh%3Ad-link%3Adir-620&do=.

[2] "Broadcom BCM5354." https://wiki.openwrt.org/toh/edimax/ps-1208mfg, 2009.

[3] "GRLIB IP Core User's Manual." https://www.gaisler.com/products/grlib/grip.pdf, 2018.

[4] M. Tehranipoor and C. Wang, *Introduction to hardware security and trust*. Springer, 2012.

[5] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors," in *ACM SIGARCH Computer Architecture News*, pp. 361–372, IEEE, 2014.

[6] V. Banciu, E. Oswald, and C. Whitnall, "Reliable information extraction for single trace attacks," in *Design, Automation and Test in Europe*, pp. 133–138, EDA Consortium, 2015.

[7] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "Scandalee: a side-channel-based disassembler using local electromagnetic emanations," in *Design, Automation and Test in Europe*, pp. 139–144, EDA Consortium, 2015.

[8] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," in *International Test Conference*, pp. 339–344, IEEE, 2004.

[9] B. Yang, K. Wu, and R. Karri, "Secure scan: A design-for-test architecture for crypto chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2287–2293, 2006.

[10] R. Nara, M. Yanagisawa, and N. Togawa, "Scan-based side-channel attack against RSA cryptosystems using scan signatures," *Fundamentals of Electronics, Communications and Computer Sciences*, vol. 93, no. 12, pp. 2481–2489, 2010.

[11] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Scan-based attack against elliptic curve cryptosystems," in *Asia and South Pacific Design Automation Conference*, pp. 407–412, IEEE, 2010.

[12] Y. Liu, K. Wu, and R. Karri, "Scan-based attacks on linear feedback shift register based stream ciphers," *Design Automation of Electronic Systems*, vol. 16, no. 2, pp. 1–15, 2011.

[13] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Are advanced DfT structures sufficient for preventing scan-attacks?," in *VLSI Test Symposium*, pp. 246–251, IEEE, 2012.

[14] J. Da Rolt, A. Das, G. Di Natale, M.-L. Flottes, B. Rouzeyre, and I. Verbauwhede, "A new scan attack on RSA in presence of industrial countermeasures," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 89–104, Springer, 2012.

[15] S. S. Ali, O. Sinanoglu, S. M. Saeed, and R. Karri, "New scan-based attack using only the test mode," in *International Conference on Very Large Scale Integration*, pp. 234–239, IEEE, 2013.

[16] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware Trojan: threats and emerging solutions," in *High Level Design Validation and Test Workshop*, pp. 166–171, IEEE, 2009.

[17] S. Adee, "The hunt for the kill switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[18] U. D. of Commerce, "Defense industrial base assessment: counterfeit electronics," 2010.

[19] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," in *Cryptographic Hardware and Embedded Systems*, pp. 363–381, Springer, 2009.

[20] E. I. Cole, "Non-destructive IC defect localization using optical beam-based imaging," in *Custom Integrated Circuits Conference*, pp. 53–56, IEEE, 2008.

[21] R. Courtland, "X-rays map the 3D interior of integrated circuits," *IEEE Spectrum*, 2017.

[22] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, 2014.

[23] M. Pecht and S. Tiku, "Bogus: electronic manufacturing and consumers confront a rising tide of counterfeit electronics," *IEEE Spectrum*, vol. 43, no. 5, pp. 37–46, 2006.

[24] "IEEE Std 1149.1-2013 - IEEE standard for test access port and boundary-scan architecture." `https://standards.ieee.org/findstds/standard/1149.1-2013.html`.

[25] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 23–40, Springer, 2012.

[26] Z. Basnight, J. Butts, J. Lopez, and T. Dube, "Firmware modification attacks on programmable logic controllers," *Critical Infrastructure Protection*, vol. 6, no. 2, pp. 76–84, 2013.

[27] I. Breeuwsma, "Forensic imaging of embedded systems using JTAG (boundary-scan)," *Digital Investigation*, vol. 3, no. 1, pp. 32–42, 2006.

[28] "LEON3 processor." `https://www.gaisler.com/index.php/products/processors/leon3`.

[29] "MinSOC." https://opencores.org/project/minsoc.

[30] "OpenSPARC T2 processor." http://www.oracle.com/technetwork/systems/opensparc/opensparc-t2-page-1446157.html.

[31] "IEEE Std 1149.4-2010 - IEEE standard for a mixed-signal test bus." https://standards.ieee.org/findstds/standard/1149.4-2010.html.

[32] "IEEE Std 1500-2005 - IEEE standard testability method for embedded core-based integrated circuits." https://standards.ieee.org/findstds/standard/1500-2005.html.

[33] "IEEE Std 1687-2014 - IEEE standard for access and control of instrumentation embedded within a semiconductor device." https://standards.ieee.org/findstds/standard/1687-2014.html.

[34] "What is JTAG?." http://www.corelis.com/education/What-Is-JTAG.htm.

[35] I. Slochinsky, "Introduction to embedded reverse engineering for PC reversers," in *REcon Conference*, p. 1, 2010.

[36] F. Domke, "Blackbox JTAG reverse engineering," in *Chaos Communication Congress*, pp. 1–5, 2009.

[37] "How to JTAG your Xbox 360 and run Homebrew." http://www.instructables.com/id/How-to-JTAG-your-Xbox-360-and-run-homebrew/.

[38] "A primer on IoT security research." https://blog.rapid7.com/2015/03/10/iot-security-research-whats-it-take/.

[39] "Why are JTAG and UART still effective attack vectors for IoT devices." https://p16.praetorian.com/blog/why-are-jtag-and-uart-still-effective-attack-vectors-for-iot-devices.

[40] M. Goryachy and M. Ermolov, *Where there's a JTAG, there's a way: obtaining full system access via USB*. Positive Technologies, Framingham, MA, USA, 2017.

[41] Intel, Santa Clara, CA, USA, *Intel hardware-based security technologies for intelligent retail devices*, 2013.

[42] Intel, Santa Clara, CA, USA, *Using the design security features in Intel FPGAs*, 2017.

[43] Altera, San Jose, CA, USA, *Protecting the FPGA design from common threats*, 2009.

[44] M. Hunter, *Using the Kinetis security and flash protection features*. NXP Semiconductors, Eindhoven, Netherlands, 2012.

[45] Silicon Labs, Austin, TX, USA, *AN0060: Bootloader with AES encryption*, 2016.

[46] "Password protected JTAG flashing query." https://www.infineonforums.com/threads/5250-Password-Protected-JTAG-Flashing-Query.

[47] Samsung ARTIK, San Jose, CA, USA, *Locking the JTAG port*, 2017.

[48] "How to enable JTAG in secure mode?." https://www.xilinx.com/support/answers/64275.html.

[49] Texas Instruments, Dallas, TX, USA, *MSP code protection features*, 2015.

[50] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," *Smartcard*, vol. 99, no. 1, pp. 9–20, 1999.

[51] "JTAGulator." http://www.grandideastudio.com/portfolio/jtagulator/.

[52] "SEGGER J-Link." https://www.segger.com/products/debug-probes/j-link/.

[53] "High quality test solutions for secure applications." https://www.mentor.com/products/silicon-yield/resources/overview/high-quality-test-solutions-for-secure-applications-1e68b08d-0490-4efb-a867-4cabba2221b5, 2013.

[54] A. Das, Ü. Kocabaş, A.-R. Sadeghi, and I. Verbauwhede, "PUF-based secure test wrapper design for cryptographic SoC testing," in *Design, Automation and Test in Europe*, pp. 866–869, EDA Consortium, 2012.

[55] S. Paul, R. S. Chakraborty, and S. Bhunia, "VIm-Scan: A low overhead scan design approach for protection of secret key in scan-based secure chips," in *VLSI Test Symposium*, pp. 455–460, IEEE, 2007.

[56] F. Novak and A. Biasizzo, "Security extension for IEEE std 1149.1," *Journal of Electronic Testing*, vol. 22, no. 3, pp. 301–303, 2006.

[57] A. Iamburg, "The JTAG interface: an attacker's perspective." https://www.optiv.com/resources/library/the-jtag-interface-an-attackers-perspective?page=1&searchQuery=&itemsPerPage=0&category=, 2016.

[58] S. Kan, J. Dworak, and J. G. Dunham, "Echeloned IJTAG data protection," in *Asian Hardware-Oriented Security and Trust*, pp. 1–6, IEEE, 2016.

[59] D. E. Denning, "An intrusion detection model," *Software Engineering*, vol. 1, no. 2, pp. 222–232, 1987.

[60] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State transition analysis: a rule-based intrusion detection approach," *Software Engineering*, vol. 21, no. 3, pp. 181–199, 1995.

[61] J. M. Estevez-Tapiador, P. Garcia-Teodoro, and J. E. Diaz-Verdejo, "Stochastic protocol modeling for anomaly based network intrusion detection," in *International Workshop on Information Assurance*, pp. 3–12, IEEE, 2003.

[62] H. Fujiwara, K. Fujiwara, and H. Tamamoto, "Secure scan design using shift register equivalents against differential behavior attack," in *Asia and South Pacific Design Automation Conference*, pp. 818–823, IEEE, 2011.

[63] J. Brownlee, "Discover feature engineering, how to engineer features and how to get good at it." https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/, 2014.

[64] "OpenSPARC T1 processor." `http://www.oracle.com/technetwork/systems/opensparc/opensparc-t1-page-1444609.html`.

[65] T. Mitchell, *Machine learning*. McGraw Hill, 1997.

[66] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[67] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[68] F. Rosenblatt, *Perceptions and the theory of brain mechanisms*. Springer, 1962.

[69] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, "Toward open set recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1757–1772, 2013.

[72] C. Bellinger, S. Sharma, and N. Japkowicz, "One-class versus binary classification: Which and when?," in *International Conference on Machine Learning and Applications*, vol. 2, pp. 102–106, IEEE, 2012.

[73] M. Inoue, T. Yoneda, M. Hasegawa, and H. Fujiwara, "Partial scan approach for secret information protection," in *European Test Symposium*, pp. 143–148, IEEE, 2009.

[74] S. M. Saeed, S. S. Ali, O. Sinanoglu, and R. Karri, "Test-mode-only scan attack and countermeasure for contemporary scan architectures," in *International Test Conference*, pp. 1–8, IEEE, 2014.

[75] J. Da Rolt and G. Di Natale, "Scan attacks and countermeasures in presence of scan response compactors," in *European Test Symposium*, pp. 19–24, IEEE, 2011.

[76] A. Das, B. Ege, S. Ghosh, L. Batina, and I. Verbauwhede, "Security analysis of industrial test compression schemes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1966–1977, 2013.

[77] D. Hely, F. Bancel, M.-L. Flottes, and B. Rouzeyre, "Secure scan techniques: a comparison," in *International On-Line Testing Symposium*, p. 6, IEEE, 2006.

[78] G.-M. Chiu and J. C.-M. Li, "A secure test wrapper design against internal and boundary scan attacks for embedded cores," *Very Large Scale Integration Systems*, vol. 20, no. 1, pp. 126–134, 2012.

[79] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing designs against scan-based side-channel attacks," *Dependable and Secure Computing*, vol. 4, no. 4, pp. 325–336, 2007.

[80] D. Hely, M.-L. Flottes, F. Bancel, B. Rouzeyre, N. Berard, and M. Renovell, "Scan design and secure chip [secure IC testing]," in *International On-Line Testing Symposium*, vol. 4, pp. 219–224, 2004.

[81] D. Zhang, M. He, X. Wang, and M. Tehranipoor, "Dynamically obfuscated scan for protecting IPs against scan-based attacks throughout supply chain," in *VLSI Test Symposium*, pp. 1–6, IEEE, 2017.

[82] J. Lee, M. Tebranipoor, and J. Plusquellic, "A low-cost solution for protecting IPs against scan-based side-channel attacks," in *VLSI Test Symposium*, p. 6, IEEE, 2006.

[83] G. Sengar, D. Mukhopadhyay, and D. R. Chowdhury, "Secured flipped scan-chain model for crypto-architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 11, pp. 2080–2084, 2007.

[84] Y. Atobe, Y. Shi, M. Yanagisawa, and N. Togawa, "Dynamically changeable secure scan architecture against scan-based side channel attack," in *International SoC Design Conference*, pp. 155–158, IEEE, 2012.

[85] M. A. Razzaq, V. Singh, and A. Singh, "SSTKR: Secure and testable scan design through test key randomization," in *Asian Test Symposium*, pp. 60–65, IEEE, 2011.

[86] "ARM AMBA specification." `https://www.arm.com/products/system-ip/amba-specifications`.

[87] "SoC interconnection: WISHBONE." `https://opencores.org/howto/wishbone`.

[88] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Annual International Cryptology Conference*, pp. 513–525, Springer, 1997.

[89] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 2–12, Springer, 2002.

[90] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel differential scan attack on advanced DFT structures," *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 4, p. 58, 2013.

[91] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A scan-based attack based on discriminators for AES cryptosystems," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 12, pp. 3229–3237, 2009.

[92] B. Ege, A. Das, L. Batina, and I. Verbauwhede, "Security of countermeasures against state-of-the-art differential scan attacks," in *European Test Symposium*, IEEE, 2013.

[93] J. Da Rolt and A. Das, "A scan-based attack on elliptic curve cryptosystems in presence of industrial design-for-testability structures," in *VLSI and Nanotechnology Systems*, pp. 43–48, IEEE, 2012.

[94] F. Majeric, B. Gonzalvo, and L. Bossuet, "JTAG combined attack: Another approach for fault injection," in *International Conference on New Technologies, Mobility and Security*, pp. 1–5, IEEE, 2016.

[95] F. Majéric, B. Gonzalvo, and L. Bossuet, "JTAG fault injection attack," *IEEE Embedded Systems Letters*, 2017.

[96] K. P. Parker, *The boundary-scan handbook*, vol. 101. Springer, 1992.

[97] H. Kodera, M. Yanagisawa, and N. Togawa, "Scan-based attack against DES cryptosystems using scan signatures," in *Asia Pacific Conference on Circuits and Systems*, pp. 599–602, IEEE, 2012.

[98] A. Ng, "Machine learning and AI via brain simulations." https://helper.ipam.ucla.edu/publications/gss2012/gss2012_10595.pdf, 2013.

[99] M. A. Hall, *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato Hamilton, 1999.

[100] P. M. Granitto, C. Furlanello, F. Biasioli, and F. Gasperi, "Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products," *Chemometrics and Intelligent Laboratory Systems*, vol. 83, no. 2, pp. 83–90, 2006.

[101] K. Grabczewski and N. Jankowski, "Feature selection with decision tree criterion," in *International Conference on Hybrid Intelligent Systems*, p. 6, IEEE, 2005.

[102] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, pp. 2579–2605, 2008.

[103] "GRMON3." `https://www.gaisler.com/index.php/products/debug-tools/grmon3`.

[104] C. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[105] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[106] J. A. Perez-Ortiz and M. L. Forcada, "Part-of-speech tagging with recurrent neural networks," in *International Joint Conference on Neural Networks*, vol. 3, pp. 1588–1592, IEEE, 2001.

[107] S. Fine, Y. Singer, and N. Tishby, "The hierarchical hidden Markov model: Analysis and applications," *Machine Learning*, vol. 32, no. 1, pp. 41–62, 1998.

[108] W. J. Scheirer, L. P. Jain, and T. E. Boult, "Probability models for open set recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2317–2324, 2014.

[109] "ROC curve." `http://www.mathworks.com/help/stats/perfcurve.html`.

[110] A. Dushistova, *Debugging with JTAG*. MontaVista Software Incorporation, San Jose, CA, USA, 2009.

[111] A. Sirotkin, "Debugging the Linux kernel with JTAG." `https://www.embedded.com/design/operating-systems/4207333/Debugging-the-Linux-kernel-with-JTAG`, 2017.

[112] Avago Technologies, San Jose, CA, USA, *Avago IJTAG implementation at the IP level*, 2012.

[113] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Securing access to reconfigurable scan networks," in *Asian Test Symposium*, pp. 295–300, IEEE, 2013.

[114] J. Dworak and A. Crouch, "Don't forget to lock your SIB: hiding instruments using P1687," in *International Test Conference*, pp. 1–10, IEEE, 2013.

[115] H. Liu and V. D. Agrawal, "Securing IEEE 1687-2014 standard instrumentation access by LFSR key," in *Asian Test Symposium*, pp. 91–96, IEEE, 2015.

[116] A. Zygmontowicz, J. Dworak, A. Crouch, and J. Potter, "Making it harder to unlock an lsib: Honeytraps and misdirection in a p1687 network," in *Design, Automation and Test in Europe*, p. 195, European Design and Automation Association, 2014.

[117] S. Gupta, J. Dworak, D. Engels, and A. Crouch, "Mitigating simple power analysis attacks on LSIB key logic," in *North Atlantic Test Workshop*, pp. 1–6, IEEE, 2017.

[118] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Fine-grained access management in reconfigurable scan networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 937–946, 2015.

[119] X. Ren, V. G. Tavares, and R. D. Blanton, "Detection of illegitimate access to JTAG via statistical learning in chip," in *Design, Automation and Test in Europe*, pp. 109–114, EDA Consortium, 2015.

[120] X. Ren, R. D. Blanton, and V. G. Tavares, "A learning-based approach to secure JTAG against unseen scan-based attacks," in *IEEE Computer Society Annual Symposium on VLSI*, pp. 541–546, IEEE, 2016.

[121] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[122] B. Bah and J. Tanner, "Improved bounds on restricted isometry constants for Gaussian matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 5, pp. 2882–2898, 2010.

[123] W. Lu, K. Kpalma, and J. Ronsin, "Sparse binary matrices of LDPC codes for compressed sensing," in *Data Compression Conference*, p. 10, 2012.

[124] R. Gallager, "Low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[125] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer, 2000.

[126] D. Lowd and C. Meek, "Adversarial learning," in *International Conference on Knowledge Discovery and Data Mining*, pp. 641–647, ACM, 2005.

[127] B. Nelson, B. Rubinstein, L. Huang, A. Joseph, S.-h. Lau, S. Lee, S. Rao, A. Tran, and D. Tygar, "Near-optimal evasion of convex-inducing classifiers," in *International Conference on Artificial Intelligence and Statistics*, pp. 549–556, 2010.

[128] J. Struharik, "Implementing decision trees in hardware," in *International Symposium on Intelligent Systems and Informatics*, pp. 41–46, IEEE, 2011.

[129] "Design Compiler." `http://www.synopsys.com/Tools/Verification/`.

[130] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[131] T. Fisher, "Ping command." `https://www.lifewire.com/ping-command-2618099`, 2018.

[132] "IoT Security." `https://www.avast.com/en-au/technology/iot-security`.

[133] J. Kaye, *BL600 firmware upgrade over JTAG*. Laird Technologies, Chesterfield, MO, USA, 2015.

[134] Struck Innovative System, Hamburg, Germany, *SIS3300/3301 JTAG firmware upgrade instructions*, 2005.

[135] Hewlett Packard Enterprise, Palo Alto, CA, USA, *HPE digitally signed firmware*, 2017.